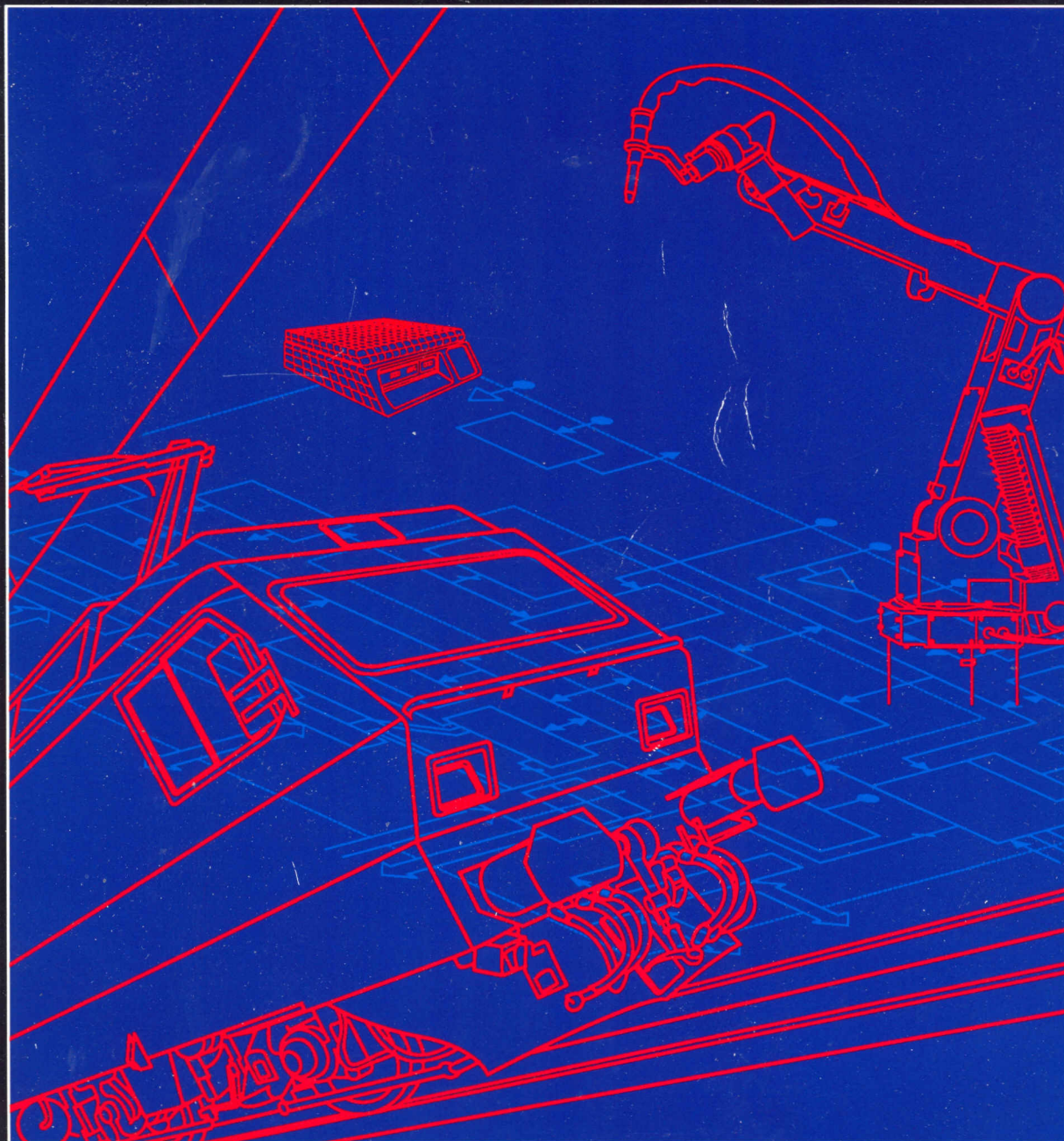




Embedded Control Applications Handbook



Order Number: 270648-001



LITERATURE

To order Intel Literature or obtain literature pricing information in the U.S. and Canada call or write Intel Literature Sales. In Europe and other international locations, please contact your local sales office or distributor.

INTEL LITERATURE SALES
P.O. BOX 58130
SANTA CLARA, CA 95052-8130

In the U.S. and Canada
call toll free
(800) 548-4725

CURRENT HANDBOOKS

Product line handbooks contain data sheets, application notes, article reprints and other design information.

TITLE	LITERATURE ORDER NUMBER
COMPLETE SET OF HANDBOOKS (Available in U.S. and Canada only)	231003
AUTOMOTIVE PRODUCTS HANDBOOK (Not included in handbook set)	231792
COMPONENTS QUALITY/RELIABILITY HANDBOOK	210997
EMBEDDED CONTROL APPLICATIONS HANDBOOK	270648
8-BIT EMBEDDED CONTROLLER HANDBOOK	270645
16-BIT EMBEDDED CONTROLLER HANDBOOK	270646
32-BIT EMBEDDED CONTROLLER HANDBOOK	270647
MEMORY COMPONENTS HANDBOOK	210830
MICROCOMMUNICATIONS HANDBOOK	231658
MICROCOMPUTER PROGRAMMABLE LOGIC HANDBOOK	296083
MICROPROCESSOR AND PERIPHERAL HANDBOOK (2 volume set)	230843
MILITARY PRODUCTS HANDBOOK (2 volume set. Not included in handbook set)	210461
OEM BOARDS AND SYSTEMS HANDBOOK	280407
PRODUCT GUIDE (Overview of Intel's complete product lines)	210846
SYSTEMS QUALITY/RELIABILITY HANDBOOK	231762
INTEL PACKAGING OUTLINES AND DIMENSIONS (Packaging types, number of leads, etc.)	231369
LITERATURE PRICE LIST (U.S. and Canada) (Comprehensive list of current Intel Literature)	210620
INTERNATIONAL LITERATURE GUIDE	E00029

CG/LIT/100188

About Our Cover:

Intel's advanced microcomputer technology for embedded control takes much of the risk out of design innovation. Our embedded controllers are able to support diverse applications as simple as weight scales and as complex as the most advanced Bullet trains. With Intel, you compose the diagrams and we'll also provide you with design tools to make those ideas happen.



Intel the Microcomputer Company:

When Intel invented the microprocessor in 1971, it created the era of microcomputers. Whether used as microcontrollers in automobiles or microwave ovens, or as personal computers or supercomputers, Intel's microcomputers have always offered leading-edge technology. In the second half of the 1980s, Intel architectures have held at least a 75% market share of microprocessors at 16 bits and above. Intel continues to strive for the highest standards in memory, microcomputer components, modules, and systems to give its customers the best possible competitive advantages.

EMBEDDED CONTROLLER APPLICATIONS HANDBOOK

The following are trademarks of Intel Corporation and may only be used to identify Intel Products:

Above, BITBUS, COMMMUTER, CREDIT, Data Pipeline, ETOX,
FASTPATH, Genius, ICE, ICEL, ICEL, IDBP, IDIS, IDICE, ILBX,
IMDIX, IMMX, Inboard, Ingate, Intel, Intel370, Intel386, Intel486,
Intel808, Intel Certified, Intelvision, Intelligent Identifier, Intelligent
Programming, Inteltek, Inteltek, IOSP, IPDS, IPSC, IRMX, ISBC,
ISBX, ISDM, ISXM, KERNOM, Library Manager, MAPNET, MCS,
Megachassis, MICROMAINFRAME, MULTIBUS, MULTICHANNEL,
MULTIMODULE, ONCE, OpenNET, OTR, PC BUBBLE, Plug-A-Bubble,
PROMPT, PROMPT, QUEST, Quest, Quick-Erase, Quick-Pulse,
Programming, Riplanmode, RMX50, RUP, Seamless, SLD, SuperCube,
UP, and VLSICE, and the combination of ICE, ICE, IRMX, ISBC, ISBX,
ISXM, MCS, or UPI and a numerical suffix, 4-SITE, 376, 386, 486.

1989

MCS is an ordering code only and is not used as a product name or trademark. MCS is a registered trademark of Microware Data Sciences Corporation.

*MULTIBUS is a patented Intel bus.

Additional copies of this manual or other Intel literature may be obtained from:

Intel Corporation
Literature Sales
P.O. Box 58130
Santa Clara, CA 95052-8130



Intel the Microcomputer Company

When Intel invented the microprocessor in 1971, it created the era of microcomputers. Whether used as microcontrollers in automobiles or microprocessors in personal computers or supercomputers, Intel's microcomputers have always offered leading-edge technology. In the second half of the 1980s, Intel's microprocessors held at least a 75% market share of microprocessors at 16 bits and above. Intel continues to strive for the highest standards in memory, microcomputer components, and systems to give its customers the best possible competitive advantage.

Intel Corporation makes no warranty for the use of its products and assumes no responsibility for any errors which may appear in this document nor does it make a commitment to update the information contained herein.

Intel retains the right to make changes to these specifications at any time, without notice.

Contact your local sales office to obtain the latest specifications before placing your order.

The following are trademarks of Intel Corporation and may only be used to identify Intel Products:

Above, BITBUS, COMMputer, CREDIT, Data Pipeline, ETOX, FASTPATH, Genius, i, i², ICE, iCEL, iCS, iDBP, iDIS, i2ICE, iLBX, i_m, iMDDX, iMMX, Inboard, Insite, Intel, int_{el}, Intel376, Intel386, Intel486, int_{el}BOS, Intel Certified, Inteleview, int_{el}gent Identifier, int_{el}gent Programming, Inteltec, Intellink, iOSP, iPDS, iPSC, iRMK, iRMX, iSBC, iSBX, iSDM, iSXM, KEPROM, Library Manager, MAPNET, MCS, Megachassis, MICROMAINFRAME, MULTIBUS, MULTICHANNEL, MULTIMODULE, ONCE, OpenNET, OTP, PC BUBBLE, Plug-A-Bubble, PROMPT, Promware, QUEST, QueX, Quick-Erase, Quick-Pulse Programming, Ripplemode, RMX/80, RUPI, Seamless, SLD, SugarCube, UPI, and VLSiCEL, and the combination of ICE, iCS, iRMX, iSBC, iSBX, iSXM, MCS, or UPI and a numerical suffix, 4-SITE, 376, 386, 486.

MDS is an ordering code only and is not used as a product name or trademark. MDS® is a registered trademark of Mohawk Data Sciences Corporation.

*MULTIBUS is a patented Intel bus.

Additional copies of this manual or other Intel literature may be obtained from:

Intel Corporation
Literature Sales
P.O. Box 58130
Santa Clara, CA 95052-8130



CUSTOMER SUPPORT

INTEL'S COMPLETE SUPPORT SOLUTION WORLDWIDE

Customer Support is Intel's complete support service that provides Intel customers with hardware support, software support, customer training, consulting services and network management services. For detailed information contact your local sales offices.

After a customer purchases any system hardware or software product, service and support become major factors in determining whether that product will continue to meet a customer's expectations. Such support requires an international support organization and a breadth of programs to meet a variety of customer needs. As you might expect, Intel's customer support is quite extensive. It can start with assistance during your development effort to network management. 100 Intel sales and service offices are located worldwide — in the U.S., Canada, Europe and the Far East. So wherever you're using Intel technology, our professional staff is within close reach.

HARDWARE SUPPORT SERVICES

Intel's hardware maintenance service, starting with complete on-site installation will boost your productivity from the start and keep you running at maximum efficiency. Support for system or board level products can be tailored to match your needs, from complete on-site repair and maintenance support economical carry-in or mail-in factory service.

Intel can provide support service for not only Intel systems and emulators, but also support for equipment in your development lab or provide service on your product to your end-user/customer.

SOFTWARE SUPPORT SERVICES

Software products are supported by our Technical Information Phone Service (TIPS) that has a special toll free number to provide you with direct, ready information on known, documented problems and deficiencies, as well as work-arounds, patches and other solutions.

Intel's software support consists of two levels of contracts. Standard support includes TIPS (Technical Information Phone Service), updates and subscription service (product-specific troubleshooting guides and; *COMMENTS Magazine*). Basic support consists of updates and the subscription service. Contracts are sold in environments which represent product groupings (e.g., iRMX® environment).

CONSULTING SERVICES

Intel provides field system engineering consulting services for any phase of your development or application effort. You can use our system engineers in a variety of ways ranging from assistance in using a new product, developing an application, personalizing training and customizing an Intel product to providing technical and management consulting. Systems Engineers are well versed in technical areas such as microcommunications, real-time applications, embedded microcontrollers, and network services. You know your application needs; we know our products. Working together we can help you get a successful product to market in the least possible time.

CUSTOMER TRAINING

Intel offers a wide range of instructional programs covering various aspects of system design and implementation. In just three to ten days a limited number of individuals learn more in a single workshop than in weeks of self-study. For optimum convenience, workshops are scheduled regularly at Training Centers worldwide or we can take our workshops to you for on-site instruction. Covering a wide variety of topics, Intel's major course categories include: architecture and assembly language, programming and operating systems, BITBUS™ and LAN applications.

NETWORK MANAGEMENT SERVICES

Today's networking products are powerful and extremely flexible. The return they can provide on your investment via increased productivity and reduced costs can be very substantial.

Intel offers complete network support, from definition of your network's physical and functional design, to implementation, installation and maintenance. Whether installing your first network or adding to an existing one, Intel's Networking Specialists can optimize network performance for you.

MCS-48 FAMILY

Chapter 1

MCS®-48 APPLICATION NOTES

AP-24 Application Techniques for the MCS®-48 Family	1-2
AP-40 Keyboard/Display Scanning with Intel's MCS®-48 Microcomputers	1-25
AP-49 Serial I/O and Math Utilities for the 8049 Microcomputers	1-50
AP-55A A High-Speed Emulator for the Intel MCS®-48 Microcomputers	1-73
AP-91 Using the 8049 as an 80 Column Printer Controller	1-173

MCS-51 FAMILY

Chapter 2

MCS®-51 APPLICATION NOTES & ARTICLE REPRINTS

AP-69 An introduction to the Intel MCS®-51 Single-Chip Microcomputer	2-1
AP-70 Using the Intel MCS®-51 Boolean Processing Capabilities	2-31
AP-223 8051 Based CRT Terminal Controller	2-76
AB-38 Interfacing the 82786 Graphic Coprocessor to the 8051	2-153
AB-39 Interfacing the Densitron LCD to the 8051	2-159
AB-40 32-Bit Math Routines for the 8051	2-166
AB-12 Designing a Mailbox Memory for Two 80C31 Microcontrollers Using EPLDs ..	2-175
AP-252 Designing with the 80C51BH	2-189
AP-410 Enhanced Serial Port on the 83C51FA	2-213
AB-41 Software Serial Port Implemented with the PCA	2-221
AP-307 83C51FA/FB PCA Cookbook	2-244
AP-425 Small DC Motor Control	2-287
AR-517 Using the 8051 Microcontroller with Resonant Transducers	2-301
AR-526 Analog/Digital Processing with Microcontrollers	2-306

Chapter 3

ASIC FAMILY APPLICATION NOTE & ARTICLE REPRINT

AP-413 Using Intel's ASIC Core Cell to Expand the Capabilities of an 80C51 Based System	3-1
AR-537 A Fast-Turnaround, Easily Testable ASIC Chip for Serial Bus Control	3-10

THE RUP1 FAMILY

Chapter 4

RUP1™ APPLICATION NOTES

AP-281 UPI-452 Accelerates 80286 Bus Performance	4-1
AP-283 RUP1™/Flexibility in Frame Size with the 8044	4-21

80186/80188 FAMILY

Chapter 5

80186/188 APPLICATION NOTES

AP-186 Using the 80186/188/C186/C188 Microprocessor	5-1
AP-258 High Speed Numerics with the 80186/80188 and 8087	5-83
AP-286 80186/188 Interface to Intel Microcontrollers	5-99
AB-36 80186/80188 DMA Latency	5-129
AB-37 80186/80188 EFI Drive and Oscillator Operation	5-132
AB-31 The 80C186/80C188 Integrated Refresh Control Unit	5-134
AB-35 DRAM Refresh/Control with the 80186/188	5-147

Table of Contents (Continued)

MCS-96 FAMILY

Chapter 6

MCS®-96 APPLICATION NOTES & ARTICLE REPRINT

AP-248 Using the 8096	6-1
AP-275 An FFT Algorithm with the MCS®-96 Products	
Including Supporting Routines and Examples	6-103
AB-32 Upgrade Path from 8096-90 to 8096BH to 80C196	6-178
AB-33 Memory Expansion for the 8096	6-181
AB-34 Integer Square Root Routine for the 8096	6-193
AP-406 MCS®-96 Analog Acquisition Primer	6-197
AR-515 A Single-Chip Image Processor	6-296

Chapter 7

MCS®-96 Diagnostic Library	7-1
----------------------------	-----

Chapter 8

80960 ARTICLE REPRINTS

AR-541 Intel's 80960: An Architecture Optimized for Embedded Control	8-1
--	-----

GENERAL MICROCONTROLLER

Chapter 9

APPLICATION NOTES

AP-125 Designing Microcontroller Systems for Electrically Noisy Environments	9-1
AP-155 Oscillators for Microcontrollers	9-23
AP-318 Intel's 87C75PF Port Expander Reduces System Size & Design Time	9-55
AP-315 Latched EPROMs Simplify Microcontroller Designs	9-80

Any of the following products may appear in this publication. If so, it must be noted that such products have counterparts manufactured by Intel Puerto Rico, Inc., Intel Puerto Rico II, Inc., and/or Intel Singapore, Ltd. The product codes/part numbers of these counterpart products are listed below next to the corresponding Intel Corporation product codes/part numbers.

Intel Corporation Product Codes/ Part Numbers	Intel Puerto Rico, Inc. Intel Puerto Rico II, Inc. Product Codes/ Part Numbers	Intel Singapore, Ltd. Product Codes/ Part Numbers	Intel Corporation Product Codes/ Part Numbers	Intel Puerto Rico, Inc. Intel Puerto Rico II, Inc. Product Codes/ Part Numbers	Intel Singapore, Ltd. Product Codes/ Part Numbers
376SKIT	p376SKIT		KM2	pKM2	
903	p903		KM4	pKM4	
904	p904		KM8	pKM8	
913	p913		KNLAN	pKNLAN	
914	p914		KT60	pKT60	
923	p923		KW140	pKW140	
924	p924		KW40	pKW40	
952	p952		KW80	pKW80	
953	p953		M1	pM1	
954	p954		M2	pM2	
ADAICE	pADAICE		M4	pM4	
B386M1	pB386M1		M8	pM8	
B386M2	pB386M2		MDS610	pMDS610	
B386M4	pB386M4		MDX3015	pMDX3015	
B386M8	pB386M8		MDX3015	pMDX3015	
C044KIT	pC044KIT		MDX3016	pMDX3016	
C252KIT	pC252KIT		MDX3016	pMDX3016	
C28	pC28		MDX457	pMDX457	
C32	pC32		MDX457	pMDX457	
C452KIT	pC452KIT		MDX458	pMDX458	
D86ASM	pD86ASM		MDX458	pMDX458	
D86C86	pD86C86		MSA96	pMSA96	
D86EDI	pD86EDI		NLAN	pNLAN	
DCM9111	pDCM9111		PCLINK		sPCLINK
DOSNET	pDOSNET		PCX344A	pPCX344A	
F1	pF1		R286ASM	pR286ASM	
GUPILOGICIID	pGUPILOGICIID		R286EDI	pR286EDI	
H4	pH4		R286PLM	pR286PLM	
I044	pI044		R286SSC	pR286SSC	
I252KIT	pI252KIT		R86FOR	pR86FOR	
I452KIT	pI452KIT		RCB4410		sRCB4410
I86ASM	pI86ASM		RCX920	pRCX920	
ICE386	pICE386		RMX286	pRMX286	
III010	pIII010		RMXNET	pRMXNET	
III086	pIII086		S301	pS301	
III086	pIII086		S386	pS386	
III111	pIII111		SBC010	pSBC010	
III186	pIII186		SBC012	pSBC012	
III186	pIII186		SBC020	pSBC020	
III198	pIII198		SBC028	pSBC028	
III212	pIII212		SBC040	pSBC040	
III286	pIII286		SBC056	pSBC056	
III286	pIII286		SBC108	pSBC108	
III515	pIII515		SBC116	pSBC116	
III520	pIII520		SBC18603	pSBC18603	
III520	pIII520		SBC186410	pSBC186410	
III531	pIII531		SBC18651	pSBC18651	
III532	pIII532		SBC186530	pSBC186530	
III533	pIII533		SBC18678	pSBC18678	
III621	pIII621		SBC18848	pSBC18848	
III707	pIII707		SBC18856	pSBC18856	
III707	pIII707		SBC208	pSBC208	
III815	pIII815		SBC214	pSBC214	
INA961	pINA961		SBC215	pSBC215	
IPAT86	pIPAT86		SBC220	pSBC220	
KAS	pKAS		SBC221	pSBC221	
KC	pKC		SBC28610	pSBC28610	
KH	pKH		SBC28612	pSBC28612	
KM1	pKM1		SBC28614	pSBC28614	

Intel Corporation Product Codes/ Part Numbers	Intel Puerto Rico, Inc. Intel Puerto Rico II, Inc. Product Codes/ Part Numbers	Intel Singapore, Ltd. Product Codes/ Part Numbers	Intel Corporation Product Codes/ Part Numbers	Intel Puerto Rico, Inc. Intel Puerto Rico II, Inc. Product Codes/ Part Numbers	Intel Singapore, Ltd. Product Codes/ Part Numbers
SBC28616	pSBC28616		SBCMEM310	pSBCMEM310	
SBC300	pSBC300		SBCMEM312	pSBCMEM312	
SBC301	pSBC301		SBCMEM320	pSBCMEM320	
SBC302	pSBC302		SBCMEM340	pSBCMEM340	
SBC304	pSBC304		SBE96	pSBE96	
SBC307	pSBC307		SBX217	pSBX217	
SBC314	pSBC314		SBX218	pSBX218	
SBC322	pSBC322		SBX270	pSBX270	
SBC324	pSBC324		SBX311	pSBX311	
SBC337	pSBC337		SBX328	pSBX328	
SBC341	pSBC341		SBX331	pSBX331	
SBC386	pSBC386	sSBC386	SBX344	pSBX344	
SBC386116	pSBC386116		SBX350	pSBX350	
SBC386120	pSBC386120		SBX351	pSBX351	
SBC38621	pSBC38621		SBX354	pSBX354	
SBC38622	pSBC38622		SBX488	pSBX488	
SBC38624	pSBC38624		SBX586		sSBX586
SBC38628	pSBC38628		SCHEMAIPLD	pSCHEMAIPLD	
SBC38631	pSBC38631		SCOM	pSCOM	
SBC38632	pSBC38632		SDK51	pSDK51	
SBC38634	pSBC38634		SDK85	pSDK85	
SBC38638	pSBC38638		SDK86	pSDK86	
SBC428	pSBC428	sSBC428	SXM217	pSXM217	
SBC464	pSBC464		SXM28612	pSXM28612	
SBC517	pSBC517		SXM386	pSXM386	
SBC519	pSBC519	sSBC519	SXM544	pSXM544	
SBC534	pSBC534	sSBC534	SXM552	pSXM552	
SBC548	pSBC548		SXM951	pSXM951	
SBC550	TSBC550		SXM955	pSXM955	
SBC550	pSBC550		SYPI20	pSYP120	
SBC550	pSBC550		SYPI301	pSYP301	
SBC552	pSBC552		SYPI302	pSYP302	
SBC556	pSBC556	sSBC556	SYPI3090	pSYP31090	
SBC569	pSBC569		SYPI311	pSYP311	
SBC589	pSBC589		SYPI3847	pSYP3847	
SBC604	pSBC604		SYR286	pSYR286	
SBC608	pSBC608		SYR86	pSYR86	
SBC614	pSBC614		SYS120	pSYS120	
SBC618	pSBC618		SYS310	pSYS310	
SBC655	pSBC655		SYS311	pSYS311	
SBC6611	pSBC6611		T60	pT60	
SBC8010	pSBC8010		TA096	pTA096	
SBC80204	pSBC80204		TA252	pTA252	
SBC8024	pSBC8024	sSBC8024	TA452	pTA452	
SBC8030	pSBC8030		W140	pW140	
SBC8605	pSBC8605	sSBC8605	W280	pW280	
SBC8612	pSBC8612		W40	pW40	
SBC8614	pSBC8614		W80	pW80	
SBC8630	pSBC8630	sSBC8630	XNX286DOC	pXNX286DOC	
SBC8635	pSBC8635	sSBC8635	XNX286DOCB	pXNX286DOCB	
SBC86C38	pSBC86C38	sSBC86C38	XNXIBASE	pXNXIBASE	
SBC8825	pSBC8825	sSBC8825	XNXIDB	pXNXIDB	
SBC8840	pSBC8840		XNXIDESK	pXNXIDESK	
SBC8845	pSBC8845	sSBC8845	XNXIPLAN	pXNXIPLAN	
SBC905	pSBC905		XNXIWORD	pXNXIWORD	
SBCLNK001	pSBCLNK001				

MCS[®]-48 Application Notes

1

processor based system. The subsystems described are analog input and output, the use of chips for function evaluation, receiving code, transmitting serial code, and parity generation. After an overview of the MCS-48 family there are discussed in a more or less independent manner.

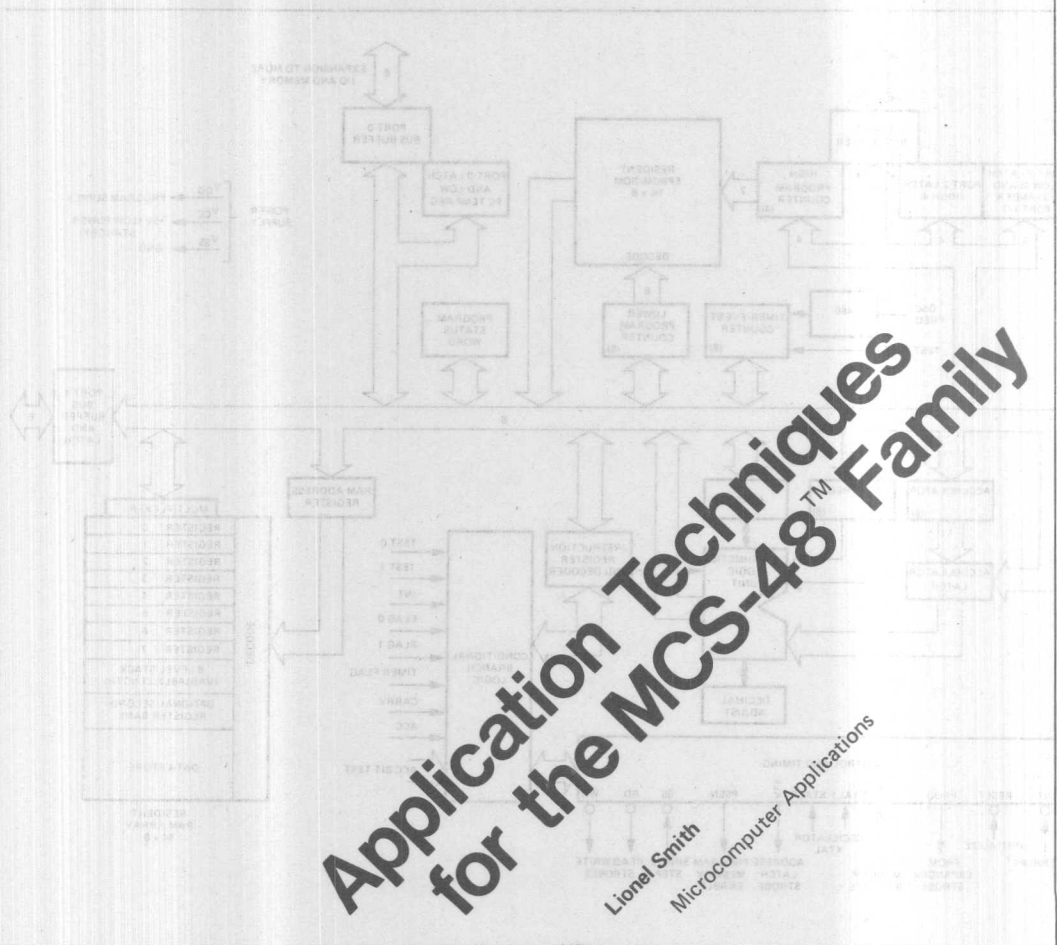
THE MCS-48™ FAMILY

The processor in the MCS-48 family all share an identical architecture. The only significant difference is the type of on-board program storage which is provided. The 8748 (see Figure 1) includes 1024 bytes of erasable programmable ROM (EPROM) the 8048 replaces the EPROM with an equivalent amount of mask programmed ROM, and the 8035 provides the CPU function with no on-board program storage. All three of these processors

INTRODUCTION
The INTEL® MCS-48™ family consists of a series of seven parts including three processors which take advantage of the advances in silicon technology to provide the system designer with an efficient solution to a wide variety of design problems. The significant contribution of the MCS-48 family is that instead of consisting of integrated micro-computer components it consists of integrated microcomputer systems. A single integrated circuit contains the processor, RAM, ROM (or PROM), a timer and I/O

February 1977

These techniques are useful with the MCS-48. Rather than presenting the design of a complete system it describes the implementation of subsystems which are common to many micro-



INTRODUCTION

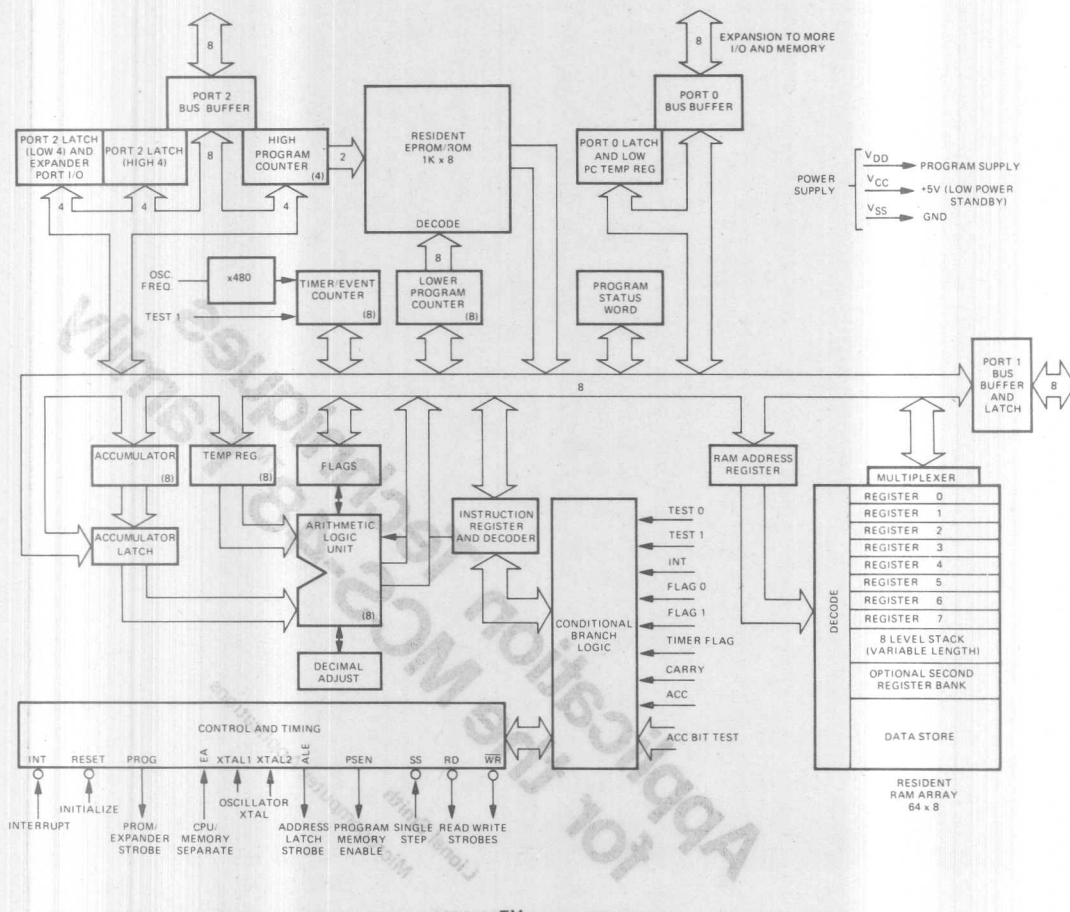
The INTEL® MCS-48™ family consists of a series of seven parts, including three processors, which take advantage of the latest advances in silicon technology to provide the system designer with an effective solution to a wide variety of design problems. The significant contribution of the MCS-48 family is that instead of consisting of integrated microcomputer components it consists of integrated microcomputer systems. A single integrated circuit contains the processor, RAM, ROM (or PROM), a timer, and I/O.

This application note suggests a variety of application techniques which are useful with the MCS-48. Rather than presenting the design of a complete system it describes the implementation of "subsystems" which are common to many micropro-

cessor based systems. The subsystems described are analog input and output, the use of tables for function evaluation, receiving serial code, transmitting serial code, and parity generation. After an overview of the MCS-48 family these areas are discussed in a more or less independent manner.

THE MCS-48™ FAMILY

The processors in the MCS-48 family all share an identical architecture. The only significant difference is the type of on board program storage which is provided. The 8748 (see Figure 1) includes 1024 bytes of erasable, programmable, ROM (EPROM), the 8048 replaces the EPROM with an equivalent amount of mask programmed ROM, and the 8035 provides the CPU function with no on board program storage. All three of these processors



MCS-48™ Internal Structure

INSTRUCTION SET

	Mnemonic	Description	Bytes	Cycle		Mnemonic	Description	Bytes	Cycles
Accumulator	ADD A,R	Add register to A	1	1	Subroutine	CALL	Jump to subroutine	2	2
	ADD A,@R	Add data memory to A	1	1		RET	Return	1	2
	ADD A,#data	Add immediate to A	2	2		RETR	Return and restore status	1	2
	ADDC A,R	Add register with carry	1	1	Flags	CLR C	Clear Carry	1	1
	ADDC A,@R	Add data memory with carry	1	1		CPL C	Complement Carry	1	1
	ADDC A,#data	Add immediate with carry	2	2		CLR F0	Clear Flag 0	1	1
	ANL A,R	And register to A	1	1		CPL F0	Complement Flag 0	1	1
	ANL A,@R	And data memory to A	1	1		CLR F1	Clear Flag 1	1	1
	ANL A,#data	And immediate to A	2	2		CPL F1	Complement Flag 1	1	1
	ORL A,R	Or register to A	1	1	Data Movers	MOV A,R	Move register to A	1	1
	ORL A,@R	Or data memory to A	1	1		MOV A,@R	Move data memory to A	1	1
	ORL A,#data	Or immediate to A	2	2		MOV A,#data	Move immediate to A	2	2
	XRL A,R	Exclusive Or register to A	1	1		MOV R,A	Move A to register	1	1
	XRL A,@R	Exclusive or data memory to A	1	1		MOV @R,A	Move A to data memory	1	1
	XRL A,#data	Exclusive or immediate to A	2	2		MOV R,#data	Move immediate to register	2	2
	INC A	Increment A	1	1		MOV @R,#data	Move immediate to data memory	2	2
	DEC A	Decrement A	1	1		MOV A,PSW	Move PSW to A	1	1
	CLR A	Clear A	1	1		MOV PSW,A	Move A to PSW	1	1
	CPL A	Complement A	1	1		XCH A,R	Exchange A and register	1	1
	DA A	Decimal Adjust A	1	1		XCH A,@R	Exchange A and data memory	1	1
	SWAP A	Swap nibbles of A	1	1		XCHD A,@R	Exchange nibble of A and register	1	1
	RL A	Rotate A left	1	1		MOVX A,@R	Move external data memory to A	1	2
	RLC A	Rotate A left through carry	1	1		MOVX @R,A	Move A to external data memory	1	2
	RR A	Rotate A right	1	1		MOVP A,@A	Move to A from current page	1	2
	RRC A	Rotate A right through carry	1	1		MOVP3 A,@A	Move to A from Page 3	1	2
Input/Output	IN A,P	Input port to A	1	2	Timer/Counter	MOV A,T	Read Timer/Counter	1	1
	OUTL P,A	Output A to port	1	2		MOV T,A	Load Timer/Counter	1	1
	ANL P,#data	And immediate to port	2	2		STRT T	Start Timer	1	1
	ORL P,#data	Or immediate to port	2	2		STRT CNT	Start Counter	1	1
	INS A,BUS	Input BUS to A	1	2		STOP TCNT	Stop Timer/Counter	1	1
	OUTL BUS,A	Output A to BUS	1	2		EN TCNTI	Enable Timer/Counter Interrupt	1	1
	ANL BUS,#data	And immediate to BUS	2	2		DIS TCNTI	Disable Timer/Counter Interrupt	1	1
	ORL BUS,#data	Or immediate to BUS	2	2	Control	EN I	Enable external interrupt	1	1
	MOVD A,P	Input Expander port to A	1	2		DIS I	Disable external interrupt	1	1
MOVD P,A	Output A to Expander port	1	2	SEL RB0		Select register bank 0	1	1	
ANLD P,A	And A to Expander port	1	2	SEL RB1		Select register bank 1	1	1	
ORLD P,A	Or A to Expander port	1	2	SEL MB0		Select memory bank 0	1	1	
Branch	INC R	Increment register	1	1		SEL MB1	Select memory bank 1	1	1
	INC @R	Increment data memory	1	1		ENTO CLK	Enable Clock output on T0	1	1
	DEC R	Decrement register	1	1		NOP	No Operation	1	1
	JMP addr	Jump unconditional	2	2	Mnemonics copyright Intel Corporation 1976				
	JMPP @A	Jump indirect	1	2					
	DJNZ R,addr	Decrement register and skip	2	2					
	JC addr	Jump on Carry = 1	2	2					
	JNC addr	Jump on Carry = 0	2	2					
	JZ addr	Jump on A Zero	2	2					
	JNZ addr	Jump on A not Zero	2	2					
	JT0 addr	Jump on T0 = 1	2	2					
	JNT0 addr	Jump on T0 = 0	2	2					
	JT1 addr	Jump on T1 = 1	2	2					
	JNT1 addr	Jump on T1 = 0	2	2					
	JF0 addr	Jump on F0 = 1	2	2					
	JF1 addr	Jump on F1 = 1	2	2					
	JTF addr	Jump on timer flag	2	2					
	JNI addr	Jump on INT = 0	2	2					
	JBb addr	Jump on Accumulator Bit	2	2					

Figure 2. 8048/8748/8035 Instruction Set

operate from a single 5-volt power supply. The 8748 requires an additional 25-volt supply only while the on board EPROM is being programmed. When installed in a system only the 5-volt supply is needed. Aside from program storage, these chips include 64 bytes of data storage (RAM), an eight bit timer which can also be used to count external events, 27 programmable I/O pins and the processor itself. The processor offers a wide range of instruction capability including many designed for bit, nibble, and byte manipulation. The instruction set is summarized in Figure 2.

Aside from the processors, the MCS-48 family includes 4 devices: one pure I/O device and 3 combination memory and I/O devices. The pure I/O device is the 8243, a device which is connected to a special 4 bit bus provided by the MCS-48 processors and which provides 16 I/O pins which can be programmatically controlled.

The combination memory and I/O devices consist of the 8355, the 8755, and the 8155. The 8355 and the 8755 both provide 2,048 bytes of program storage and two eight bit data ports. The only difference between these devices is that the 8355 contains masked program ROM and the 8755 contains EPROM. The 8155 combines 256 bytes of data storage (RAM), two eight bit data ports, a six bit control port, and a 14 bit programmable timer.

Figure 3 shows the various system configurations which can be achieved using the MCS-48 family of parts. It should also be noted that eight of the processors' I/O lines have been configured as a bidirectional bus which can be used to interface to standard Intel peripheral parts such as the 8251 USART (for serial I/O), the 8255A PPI (provides 24 I/O lines) and the complete range of memory components.

More detailed information concerning the MCS-48 family can be obtained from the "MCS-48 Microcomputer User's Manual" which provides a complete description of the MCS-48 family and its members. A general familiarity with this document will make the application techniques which follow easier to understand.

ANALOG I/O

If analog I/O is required for a MCS-48™ system there are many alternatives available from the makers of analog I/O modules. By searching through their catalogs it is possible to find almost any combination of features which is technically feasible. Perhaps the best example of such modules are the MP-10 and MP-20 hybrid modules recently introduced by Burr-Brown Research Corporation. The MP-10 provides two analog outputs and the MP-20 provides 16 analog inputs. Both of these units were

[] Number of Available Timers
() Number of Available I/O Lines

DATA MEMORY (RAM)	PROGRAM MEMORY (ROM)			
	1K	2K	3K	4K
1088	8048 4-8155 [5] (101)	8035 8355 4-8155 [5] (116)	8048 8355 4-8155 [5] (116)	8035 2-8355 4-8155 [5] (131)
832	8048 3-8155 [4] (80)	8035 8355 3-8155 [4] (95)	8048 8355 3-8155 [4] (95)	8035 2-8355 3-8155 [4] (110)
578	8048 2-8155 [3] (59)	8035 8355 2-8155 [3] (74)	8048 8355 2-8155 [3] (74)	8035 2-8355 2-8155 [3] (89)
512	8048 8155 [2] (38)	8035 8355 8155 [2] (53)	8048 8355 8155 [2] (53)	8035 2-8355 8155 [2] (68)
320	8048 [1] (24)	8035 8355 [1] (28)	8048 8355 [1] (28)	8035 2-8355 [1] (43)
256				
64				

Figure 3. The Expanded MCS-48™ System

specifically designed to interface with microprocessors.

A block diagram of the MP-10 is shown in Figure 4. It consists of two eight bit digital to analog converters, two eight bit latches which are loaded from the data bus, and address decoding logic to determine when the latches should be loaded. The D/A converters each generate an analog output in the range of 10 volts with an output impedance of 1Ω. Accuracy is ±0.4% of full scale and the output is stable 25μsec after the eight bit binary data is loaded into the appropriate latch. The latches are loaded by the write pulse (WR) whenever the proper address is presented to the MP-10. The lower two addresses (A₀ and A₁) are used internally by the device. Addresses A₂ & A₃ are compared with the address determination inputs B₂ and B₃. If their signals are found to be equal, and if addresses A₄-A₁₃ are all high, then the device is selected and one of the latches will be loaded. Address bit A₁ selects between output 1 and output 2. If address bit A₀ is set then the initialization channel of the DIA is selected. In order to prepare for operation a data pattern of 80H must

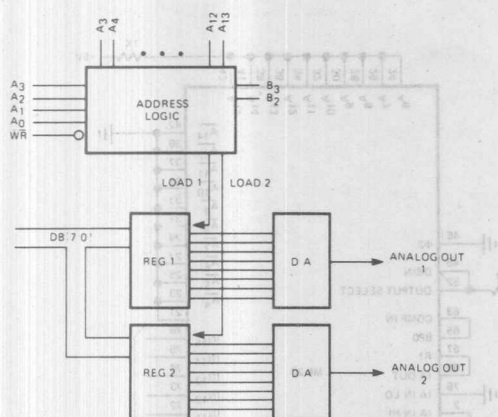


Figure 4. MP-10 Block Diagram

be output to this channel following the reset of the device.

A block diagram of the MP-20 analog to digital converter is shown in figure 5. This unit consists of a 16 input analog multiplexer, an instrumentation amplifier, an eight bit successive approximation analog to digital converter, and control logic. The 16 input multiplexer can be used to input either 16 single ended or 8 differential inputs. The output from the multiplexer is fed into the instrumentation amplifier which is configured so that it can easily be strapped for single ended 0-5 volt inputs, single ended ± 5 volt inputs, or differential 0-5 volt signals. Provisions are made for an external gain control resistor on the amplifier. The gain control equation is:

$$G = 2 + \frac{50k\Omega}{R_{ext}}$$

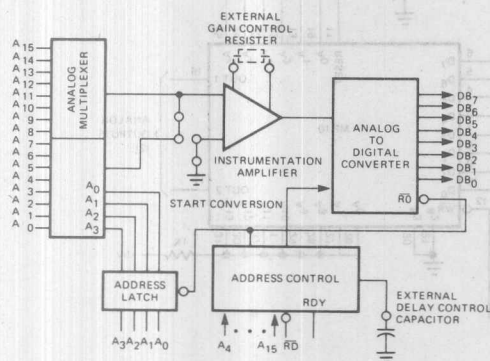


Figure 5. MP-20 Analog Subsystem

With no R_{ext} ($R_{ext} = \infty$) the gain is two and the input is 0-5 or ± 5 volts full scale. Adding an external resistor results in higher gain so that low level ($\pm 50mV$) signals from thermocouples and strain gauges can be accommodated. The output from the amplifier is applied to the actual A/D converter which provides an eight bit output with guaranteed monotonicity and an accuracy of $\pm 0.4\%$ of full scale. Note that this accuracy is specified for the entire module, not just for the converter itself. The control logic monitors address lines A15 through A4 to determine when the address of the unit has been selected. An address that the unit will respond to is determined by 11 address control pins, labeled A4 through A14. If one of these pins is tied to a logic 0 then the corresponding address pin must be high in order for the unit to be selected. If the pin is tied to a logic 1 then the corresponding address pin must be low. If the address of the module is selected when MEMR pulse occurs, the lower four addresses (A3-A0) are stored in a latch which addresses the multiplexer. The coincidence of the proper address and MEMR also initiates a conversion and gates the output of the converter on to the eight bit data bus.

The control logic of the MP-20 was designed to operate directly with an MCS-80™ system. When a MEMR occurs and a conversion is initiated the MP-20 generates a READY signal which is used to extend the cycle of the 8080A for the duration of the conversion. READY is brought high after the conversion is complete which allows the 8080A to initiate a conversion and read the resulting data in a single, albeit long, memory or I/O cycle. The conversion time of the MP-20 depends on the gain selected for the amplifier. With no external resistor ($R = \infty$) the gain is two and the conversion time is 35 μsec . For $R = 510\Omega$ the gain is:

$$G = 2 + \frac{50k\Omega}{.51k\Omega} \approx 100$$

and the conversion time becomes 100 μsec . These settling times are specified in the MP-20 data sheet and range from 35 to 175 microseconds. The READY timing is controlled by an external capacitor. For a gain of 2 no external capacitor is required but if higher gains are selected a capacitor is needed to extend the timing.

A schematic showing both the MP-10 D/A and the MP-20 A/D connected to the 8748 is shown in Figure 6. This configuration, which consists of only four major components, gives an excellent example of what modern technology can do for

the system designer. The four components provide:

- An eight bit microprocessor
- 64 bytes of RAM
- 1024 bytes of UV erasable PROM
- A timer/event counter
- 16 digital I/O pins
- 2 testable input pins
- An interrupt capability
- 16 eight bit analog inputs
- 2 eight bit analog outputs

The MCS-48 communicates with the D/A and A/D converters in a memory mapped mode (i.e., it treats the devices as if they were external RAM). By setting an address in either R0 or R1 and then executing a MOVX the software can transfer data between the accumulator and the analog I/O. When the MCS-48 executes the MOVX instruction it first sends the eight bit address out on the bus and strobes it into the 8212 latch with the ALE (Address Latch Enable) signal. After the address is latched, the MCS-48 uses the same bus to transfer data to or from the accumulator. If data is being sent out (MOVX @Rj, A) the WR strobe is used; if the data is being moved into the accumulator (MOVX A, @Rj) the RD strobe is used. The one shots on the WR line are used to delay the write strobe of the MCS-48 to meet the data set up specifications of the MP-10.

In order to provide reset capability for the analog devices without dedicating an I/O pin from the MCS-48, special addresses are used as reset channels. Executing any MOVX with an address of 0XXXXXXX will reset the A/D module; a similar operation with an address of X1XXXXXX will reset the D/A; a MOVX with an address of 01XXXXXX will reset both devices. All data transfers are accomplished with the upper two bits of the address field equal to 10. A summary of the addressing of the analog devices is shown in Table 1. Notice that except for an initialization channel for the D/A (which must

Table 1. Analog Interface Addresses

INPUT OR OUTPUT		
0XXX XXXX	Reset A/D	
X1XX XXXX	Reset D/A	
INPUT		
0011 nnnn	Read A/D Channel nnnn	
OUTPUT		
1011 0001	Initialize D/A	
1011 0000	Write Channel 1	
1011 0010	Write Channel 2	

be written to following a reset to initialize its internal logic) all channels involve some form of data transfer.

As was mentioned previously, the MP-20 was designed to use the READY line of the 8080A. Obviously this presents a problem since the MCS-48 does not support a READY line (with its attendant requirement of entering WAIT state). The necessity of a READY input can be overcome by performing a read operation to set the channel address, waiting the required delay (35 μ sec for a gain of two) and then performing a second read to actually obtain the data. The second read will read in the data from the channel selected by the first read irrespective of the channel selected for the second read. Thus it is possible to use the second read to set up the channel for the third read. Each read can read in the current channel and select the next channel for conversion.

The MP-20 is shown in Figure 6 strapped to input 16 single ended ± 5 volts signals. Programs which were used to test this configuration are shown in Figure 7. The first of these programs uses the D/A converter to generate sawtooth waveforms by outputting an incrementing value to the D/A converters. The second program scans the analog inputs and stores their digital values in a table located in RAM.

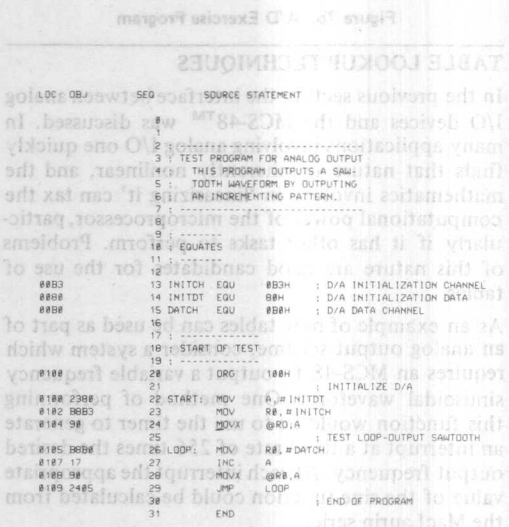


Figure 7a. D/A Exercise Program


```

LOC OBJ      SEQ      SOURCE STATEMENT
0
1
2
3 : TEST PROGRAM FOR ANALOG INPUT
4 : THIS PROGRAM SCANS THE INPUT CHANNELS
5 : AND STORES THE READINGS IN A TABLE
6 : STARTING AT BUFF.
7 :
8 :
9 :
10 : EQUATES
11 :
12 :
13 BUFF EQU 2BH ; START OF BUFFER
14 MAXCH EQU 15 ; NO OF ANALOG INPUTS
15 AINCH EQU 8BH ; BASE ADDRESS OF ANALOG INPUTS
16 TICK EQU 5 ; EXECUTION TIME OF DJNZ
17 :
18 :
19 : START OF TEST
20 :
21 ORG 100H
22
23 START: MOV R1, #BUFF-MAXCH
24         MOV R3, #MAXCH
25         MOV R8, #(AINCH-MAXCH)
26         ; SELECT CHANNEL 15
27         MOVX A, @R8
28         ; WAIT >48 MICROSECONDS
29         MOV R4, #48/TICK
30         DJNZ R4, $
31         ; NOW SCAN ANALOGS
32 LOOP:  DEC R8
33         ; GET DATA
34         MOVX A, @R8
35         ; MOVE INTO BUFFER
36         MOV @R1, A
37         ; DECREMENT BUFFER POINT
38         DEC R1
39         ; PAD 28 MICROSEC
40         MOV R4, #28/TICK
41         DJNZ R4, $
42         ; LOOP UNTIL DONE
43         DJNZ R3, LOOP
44         ; REPEAT TEST FOREVER
45         JMP START
46         ; END OF PROGRAM
47         END

```

Figure 7b. A/D Exercise Program

TABLE LOOKUP TECHNIQUES

In the previous section the interface between analog I/O devices and the MCS-48™ was discussed. In many applications involving analog I/O one quickly finds that nature is inherently nonlinear, and the mathematics involved in 'linearizing it' can tax the computational power of the microprocessor, particularly if it has other tasks to perform. Problems of this nature are good candidates for the use of tables.

As an example of how tables can be used as part of an analog output scheme, consider a system which requires an MCS-48 to output a variable frequency sinusoidal waveform. One method of performing this function would be to use the timer to generate an interrupt at a fixed rate of 256 times the desired output frequency. At each interrupt the appropriate value of the sine function could be calculated from the MacLaurin series:

$$\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} \dots \frac{(-1)^k x^{2k+1}}{(2K+1)!}$$

Where K is chosen to be large enough to provide the required accuracy.

All mnemonics copyrighted © Intel Corporation 1976.

The above calculation, although conceptually simple, would be time consuming and would severely limit the possible output frequencies which could be obtained. As an alternative to calculating these values in real time, the values could be precalculated off line and stored in a table. Upon each interrupt the MCS-48 would merely have to retrieve the appropriate value from the table and output it to the D/A converter. The MCS-48 provides a special instruction which can be used to access data in a table. If the table is stored in the last 256 bytes of the first kilobyte of MCS-48 memory then the table lookup can be performed by loading the independent variable (time in this case) into the accumulator and executing the instruction.

MOVP3 A, @A

This instruction uses the initial contents of the accumulator to index into page 3 of program storage. The location pointed to is read and the contents placed in the accumulator. If (as is often the case) a table of fewer than 256 entries is required, then the table can be located in any page of program memory and the instruction:

MOVP A, @A

can be used to retrieve data from the table. This instruction operates in the same manner as does the previous instruction except that the current page of program storage is assumed to contain the table.

If it is possible to devote slightly more of the microprocessor's time to the table look up process, then a much smaller table can often be utilized by taking advantage of interpolation to determine values of the function between values which are actual entries in the table. As an example of this

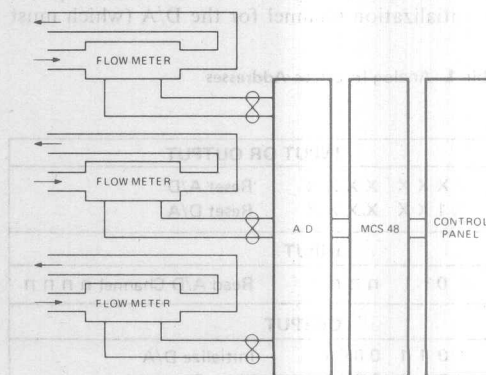


Figure 8. Flow Monitoring System

process consider the hypothetical system shown in Figure 8. The purpose of this system is to measure the flow through the three pipes, add them, and display the total flow on the control panel. The system consists of three flow meters which generate a differential voltage which is some function of flow, an A/D system with at least three differential inputs, an MCS-48, and a control panel. The schematic shown in Figure 6 could easily become part of this system, with the spare digital I/O of the MCS-48 used as an interface to the control panel. The simplicity of this system is clouded by the flow transducers, which are assumed to be not only nonlinear but also to require individual calibration (this is not an unreasonable assumption for a flow transducer). By using a table look up process and an 8748 the flow transducers can be calibrated and the results of the calibration tests stored directly in tables in the 8748. (The 8748 has a PROM in place of the ROM of the 8048 and thus makes such 'one off' programming practical.)

The results which might be obtained from calibrating one of the flow meters is shown in Figure 9. The results are plotted as gals/hour versus the measured voltage generated by the transducer. The voltage is shown in hexadecimal form so that it corresponds directly to the digital output of the analog to digital converter. The flow required to generate seventeen evenly spaced voltages (00H-100H in steps of 10H) has been measured and plotted. This information is shown in tabular form in Figure 10. It is necessary to generate a program which will convert any measured input from 00H to FFH into the flow in units which can be interpreted by a human operator. This can easily be done by simple interpolation.

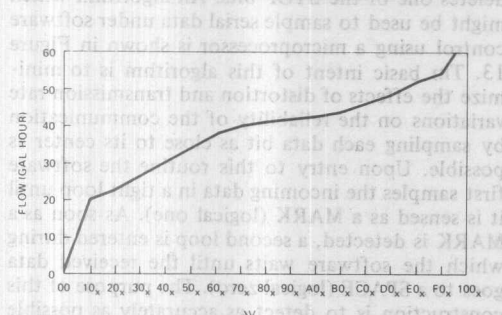


Figure 9. Flow Calibration Curve

TRANSDUCER VOLTAGE (HEX)	00	10	20	30	40	50	60	70	80	90	A0	B0	C0	D0	E0	F0	100
MEASURED FLOW (GAL HOUR)	0	10	22	26	30	34	38	40	41	42	43	45	48	49	53	56	63

Figure 10. Tabulated Flow Data

The eight bits of independent variable (voltage) can be looked on as two four bit fields. The most significant four bits (7-4) will be used to retrieve one of the table values. The lower four bits (3-0) will be used to interpolate between this value and the value retrieved from the next higher location in the table. If the upper four bits are given the symbol I and the lower four bits the symbol N, then the interpolation can be expressed as:

$$F(x) = F(I) + \frac{N}{16} [F(I+1) - F(I)]$$

Where x is the measured voltage and F(x) is the corresponding flow.

If, as an example, the transducer voltage was measured as 48H then the flow (ref. Figure 10) would be:

$$F = 30 + \frac{8}{16} (34 - 30) = 32$$

A subroutine which implements this calculation is shown in Figure 11. Before it is called the independent variable (V) is placed in the accumulator and register R1 is set to point at the first value in the table. Aside from simple additions and subtractions the only arithmetic required is to multiply two values and then divide them by 16. The multiplication is handled via a subroutine which is also shown in Figure 11. The division by 16 can be performed by a four place right shift followed by a rounding operation. The routine shown will handle a monotonic increasing function of a single independent variable. Fairly simple modifications are required for nonmonotonic functions. Functions of two variables can be handled by interpolating on a plane rather than along a straight line. Although this is more time consuming, requiring an interpolation for each of the independent variables and a third to interpolate the final answer, it still provides a simple means of quickly calculating the required function. The use of tables can offer a powerful technique for function evaluation to the designer.

RECEIVING SERIAL CODE—BASIC APPROACHES

Many microprocessor based systems require some form of serial communication. Serial communication is extensively used because it allows two or more pieces of equipment to exchange information with a minimal number of interconnecting wires. The minimization of interconnecting wires results in simpler, cheaper, interconnects because fewer (or smaller) cables and connectors are required. Since the required number of drivers and receivers required is reduced, it can become economically feasible to provide much higher noise immunity

LOC	OBJ	SEQ	SOURCE STATEMENT	LOC	OBJ	SEQ	SOURCE STATEMENT
0		1	APPROX	011C 03		56	RET
1		2	AT ENTRY R1 POINTS AT TABLE	57		57	
2		3	A HAS INDEPENDENT VARIABLE	58		58	
3		4		59		59	
4		5		60		60	MULTIPLY
5		6		61		61	
6		7		62		62	SET UP COUNT AND AEX
7		8		63		63	MULT: MOV COUNT, #8
8		9	EQUATES	64		64	MOV AEX, #8
9		10		65		65	CLEAR CARRY
10		11		66		66	LOOPA: CLR C
11		12	R1 EQU R0: POINTER #1	67		67	IF MULTIPLIER (8) < 1 THEN SHIFT PRODUCT
12		13	R1 EQU R1: POINTER1	68		68	LOOPB: JBN SSUM
13		14	AEX EQU R2: EXTENSION OF A REGISTER	69		69	XCH A, AEX
14		15	COUNT EQU R3: COUNTER	70		70	XCH A, AEX
15		16	TEMP EQU R4: TEMP STORAGE	71		71	XCH A, AEX
16		17		72		72	RRC A
17		18		73		73	LOOP UNTIL DONE
18		19	APPROXIMATION	74		74	DJNZ COUNT, LOOPB
19		20		75		75	RET
20		21		76		76	ELSE ADD MULTIPLIER AND SHIFT PRODUCT
21		22	ORG 100H	77		77	SSUM: XCH A, AEX
22		23		78		78	ADD A, @RX8
23		24	APPROX: MOV RX8, #TEMP	79		79	RRC A
24		25		80		80	XCH A, AEX
25		26	TEMP=N AND 8FH	81		81	RRC A
26		27		82		82	LOOP UNTIL DONE
27		28	MOV @RX8, #0	83		83	DJNZ COUNT, LOOPA
28		29	XCHD A, @RX8	84		84	RET
29		30	SWAP A	85		85	
30		31		86		86	
31		32	ADD A, RX1	87		87	
32		33	MOV RX1, A	88		88	TABLE TO TEST PROGRAM
33		34		89		89	
34		35	MOV P3, A, @A	90		90	
35		36	XCH A, RX1	91		91	
36		37	INC A	92		92	ORG 300H
37		38	MOV P3, A, @A	93		93	TABLE: DB 00
38		39		94		94	DB 10
39		40	CPL A	95		95	DB 22
40		41	ADD A, RX1	96		96	DB 26
41		42	CPL A	97		97	DB 38
42		43		98		98	DB 34
43		44	MULT: A=N*A/Y	99		99	DB 30
44		45	CALL MOV, RX8, #AEX	100		100	DB 48
45		46	XCHD A, @RX8	101		101	DB 41
46		47	SWAP A	102		102	DB 42
47		48	XCH A, AEX	103		103	DB 43
48		49	JB3 ADJUST	104		104	DB 45
49		50	XCH A, AEX	105		105	DB 48
50		51	ADJUST: XCH A, AEX	106		106	DB 49
51		52	INC A	107		107	DB 53
52		53		108		108	DB 56
53		54	ADD A, A-TABLE(P)	109		109	DB 63
54		55		110		110	
55		56	ADD A, RX1	111		111	END
56		57	RETURN				

Figure 11. Table Lookup With Interpolation

with more sophisticated (and expensive) line terminators. The final, and usually most persuasive, argument in favor of serial communication is that it may be the only method available to accomplish the job. The obvious example of this is telecommunications where it is necessary to encode parallel information into serial format in order to communicate via the telephone network. The intent of this section is to show how the facilities of the MCS-48™ can be brought to bear on the problem of serial communication.

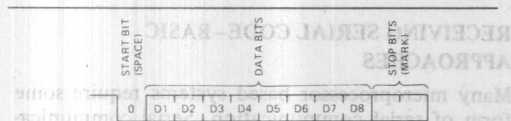


Figure 12. Serial ASCII Code

Probably the most common form of serial communication is that used by the ubiquitous Teletype—serial ASCII. This format, shown in Figure 12, consists of a START bit (0 or SPACE) followed by eight data bits which are in turn followed by two STOP bits (1 or MARK). In actual practice the

eight data bit usually consists of even parity on the remaining seven data bits; for the purposes of this discussion the eighth bit will be considered only as data. A minor variation of this format deletes one of the STOP bits. An algorithm which might be used to sample serial data under software control using a microprocessor is shown in Figure 13. The basic intent of this algorithm is to minimize the effects of distortion and transmission rate variations on the reliability of the communication by sampling each data bit as close to its center as possible. Upon entry to this routine the software first samples the incoming data in a tight loop until it is sensed as a MARK (logical one). As soon as a MARK is detected, a second loop is entered during which the software waits until the received data goes to a SPACE (logical zero). The purpose of this construction is to detect as accurately as possible the leading edge of the START bit. This instant of time will be used as a reference point for sampling all of the following bits in the character. After sensing the leading edge of the START bit a wait of one half the expected bit time is implemented. The period of the incoming signal is called P for convenience. At the end of this wait the serial line is tested—if it is MARK then the START bit was

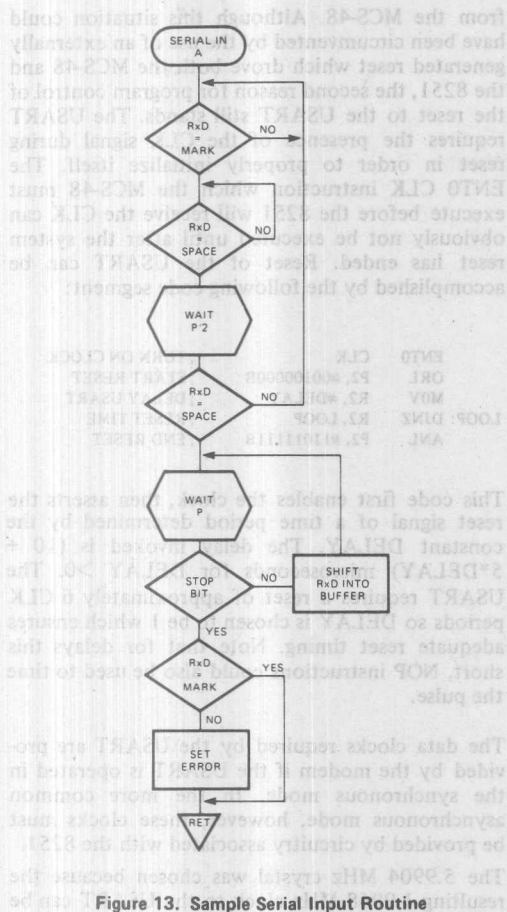


Figure 13. Sample Serial Input Routine

invalid and the process is reinitialized. If the line is still a SPACE, then the START bit is assumed to be valid and a delay of one bit time is started. At the completion of the delay the first data bit is sampled and a new delay of one bit time is initiated. This process is repeated until all eight data bits have been sampled. The last bit sampled is checked to determine if it is a valid STOP bit (a MARK). If it is, the character is assumed to be valid; if it is not, the character has a framing error and is probably invalid. A listing of a program which implements the above procedure is shown in Figure 14.

A disadvantage of the approach outlined in Figure 13 is that while the processor is inputting data serially it must totally dedicate itself to this task. Accurate timing can only be maintained if the program remains in a tight wait loop without allowing itself to be diverted to other functions. During reception of a character from a Teletype

the processor will spend only a 100µsecs or so processing data and the rest of the 100 millisecs waiting to do the processing at the right time. This lack of efficiency (approximately 0.1%) in the utilization of processing power is why devices such as the 8251 USART find broad application in micro-processor systems.

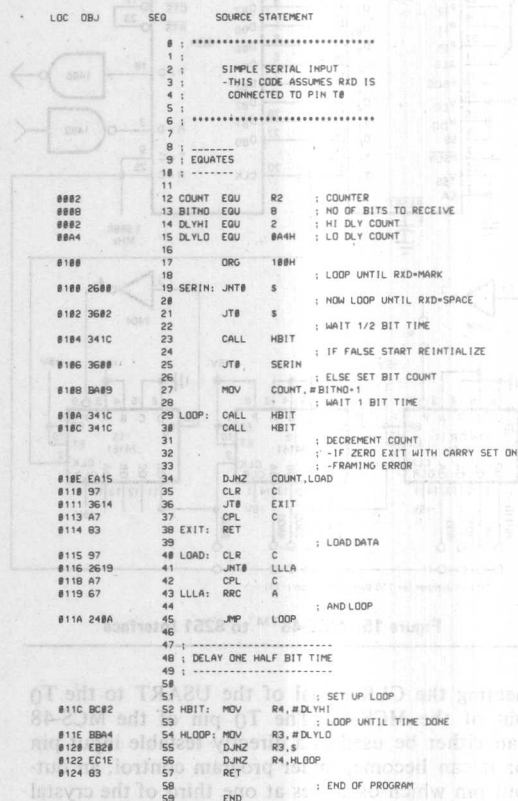
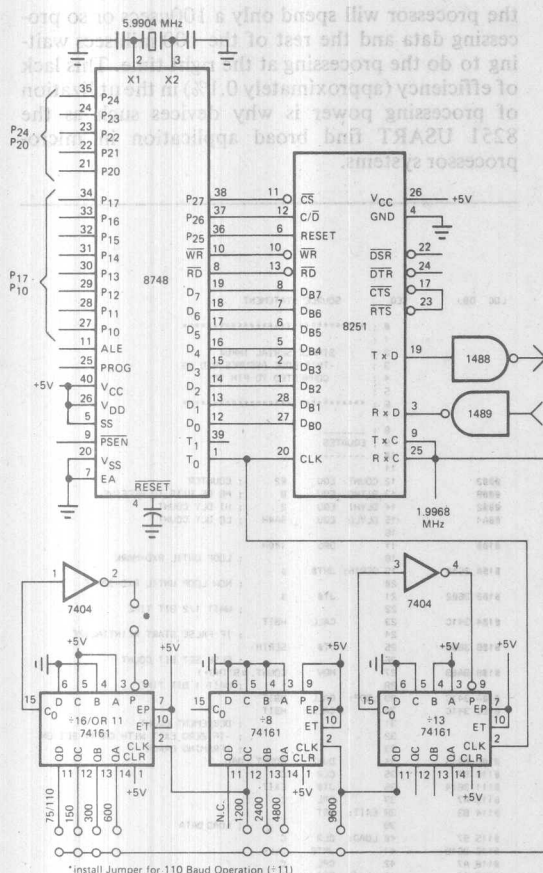


Figure 14. Simple Serial Input

The 8251 USART is simple to interface to the MSC-48. Figure 15 shows such an interface. The USART requires a high speed clock (CLK), an initialization signal (RESET), data clocks (TxC and RxC), and data in order to operate. A circuit showing the connection of an 8748 to an 8251 USART is shown in Figure 15. In the circuit shown the high speed clock (which is used for internal sequencing by the USART) is provided by con-



necting the CLK signal of the USART to the T₀ pin of the MCS-48. The T₀ pin of the MCS-48 can either be used as a directly testable input pin or it can become, under program control, an output pin which oscillates at one third of the crystal frequency. (Note that once this pin is designated by the software to be an output it will remain so until the system is reset.) In Figure 15 the crystal frequency is 5.9904 MHz so the clock provided to the 8251 is 1.9968 MHz, which conforms to its specifications.

The initialization signal to the USART (RESET) is provided programmatically by manipulation of bit 5 of port 2. It was necessary to place the reset of the 8251 under program control for two reasons. The first reason is that the MCS-48 does not supply a reset signal to other devices. The reason for this is that it was felt to be more useful to provide another pin of I/O function instead of a RESET OUT signal

from the MCS-48. Although this situation could have been circumvented by the use of an externally generated reset which drove both the MCS-48 and the 8251, the second reason for program control of the reset to the USART still stands. The USART requires the presence of the CLK signal during reset in order to properly initialize itself. The ENT0 CLK instruction which the MCS-48 must execute before the 8251 will receive the CLK can obviously not be executed until after the system reset has ended. Reset of the USART can be accomplished by the following code segment:

```

        ENT0    CLK                ; TURN ON CLOCK
        ORL     P2, #00100000B    ; START RESET
        MOV     R2, #DELAY        ; DELAY USART
LOOP:    DJNZ   R2, LOOP          ; RESET TIME
        ANL     P2, #11011111B    ; END RESET

```

This code first enables the clock, then asserts the reset signal of a time period determined by the constant DELAY. The delay invoked is (10 + 5*DELAY) microseconds for DELAY > 0. The USART requires a reset of approximately 6 CLK periods so DELAY is chosen to be 1 which ensures adequate reset timing. Note that for delays this short, NOP instructions could also be used to time the pulse.

The data clocks required by the USART are provided by the modem if the USART is operated in the synchronous mode. In the more common asynchronous mode, however, these clocks must be provided by circuitry associated with the 8251.

The 5.9904 MHz crystal was chosen because the resulting 1.9968 MHz clock to the USART can be evenly divided to provide transmit and receive clocks to the USART. Assuming the USART is in the x16 mode (i.e. it requires data clocks 16 times the baud rate) the 1.9968 MHz signal can be divided by 13 to generate the proper clock rate for 9600 baud operation. This 9600 baud clock can be further divided to give 4800, 2400, 1200, 600, and 300 baud signals. The 1200 baud signal can be divided by 11 to give a 109.1 baud signal which is within 1% of the 110 baud required by Teletypes.

The MCS-48 communicates with the 8251 in a memory mapped mode (i.e. as if the 8251 were external RAM). The instructions available to do this are MOVX @Rj, A which stores the contents of the accumulator at the external RAM location addressed by Rj (j=0 or 1), and its complement, the MOVX A, @ Rj instruction which moves data from the external RAM into the accumulator. Since the MCS-48 multiplexes addresses and data on the same eight bit bus an external latch would be required in order to address the USART with

Figure 16. 8251 Test Program

All mnemonics copyrighted © Intel Corporation 1976.

The diagram shows the following connections:

- 8748 P27 (WR)** to **8251 CS**
- 8748 P26 (RD)** to **8251 C/D**
- 8748 ALE** to **8251 D7**
- 8748 D7** to **8251 MD**
- 8748 D6** to **8251 DS2**
- 8748 D5** to **8251 D5**
- 8748 D4** to **8251 D4**
- 8748 D3** to **8251 D3**
- 8748 D2** to **8251 D2**
- 8748 D1** to **8251 D1**
- 8748 D0** to **8251 D0**

Figure 17. Modified MCS-48 to 8251 Interface

1-13

```

LOC  OBJ  SEQ  SOURCE STATEMENT
0  SERIAL TEST
1  THIS CODE INITIALIZES THE USART
2  AND TRANSMITS AN INCREMENTING
3  PATTERN. HARDWARE SHOWN IF FIG 17.
4  -----
5  -----
6  -----
7  -----
8  EQUATES
9  -----
10 -----
11 MCLR EQU 20H : USART RESET ADDRESS
12 DLY EQU 01H : USART RESET DELAY
13 UCON EQU 03H : USART CONTROL ADDRESS
14 MODE EQU 0CEH : USART MODE
15 CHD EQU 21H : USART CHD
16 STAT EQU 03H : USART STATUS
17 VAL EQU R1 : TEST VALUE
18 DATA EQU 00 : USART DATA ADDRESS
19 -----
20 ORG 100H
21 : TURN ON CLOCK
22 : AND RESET USART
23 TEST: ENTB CLK
24 ORL P2, #MCLR
25 MOV R2, #DLY
26 LOOP DJNZ R2, LOOP
27 ANL P2, #NOT MCLR
28 : SELECT USART CONTROL
29 MOV A, #UCON
30 : SEND MODE AND COMMAND
31 MOV A, #MODE
32 MOVX @R0, A : (CONTENTS OF R0 UNIMPORTANT)
33 MOV A, #CHD
34 MOVX @R0, A
35 : DO FOREVER
36 : SELECT USART STATUS
37 : IF TARDY+1 THEN
38 : DO:
39 : OUTPUT VALUE;
40 : INCREMENT VALUE;
41 : END;
42 : END;
43 TLP: MOV A, #STAT
44 MOVX @R0, A : (CONTENTS OF R0 UNIMPORTANT)
45 RRC A
46 JNC TLP
47 MOV A, VAL
48 MOV R0, #DATA
49 MOVX @R0, A
50 INC VAL
51 JMP TLP
52 : END OF PROGRAM
53 END

```

Figure 18. Modified 8251 Test Program

The timer resident on the MCS-48 provides a solution to this problem. Instead of spinning in a loop the program can set the timer for a given interval, start it, and proceed to other tasks. When the timer overflows, an interrupt will be generated to notify the software that the present time period has elapsed. An extension of the algorithm of Figure 13 which uses the timer in this fashion is shown in Figure 19. This algorithm is identical to the preceding one up until the detection of the leading edge of the start bit. At this point the timer is set to one half of the bit time (P) and a return is made to the calling program which can start additional processing. At the completion of this time interval a timer overflow interrupt is generated. When the first interrupt is detected, the serial line is checked to ensure that it is in a spacing condition (valid START bit). If it is, the timer is set to P (to sample the middle of the first data bit) and a return is made to the program which was running when the

All mnemonics copyrighted © Intel Corporation 1976.

interrupt occurred. If the serial line has returned to the MARK state, a status flag is set to indicate an error and a return is made. On subsequent interrupt detection, the data is sampled, the timer is reinitiated, and control is returned to the program which was running when the interrupt occurred. When the last (i.e. STOP) bit is detected a completion flag is set and a return is made to the program running when the timer overflow occurred. By periodically checking the error and completion flags the running program can determine when the interrupt driven receive program has a character assembled for it.

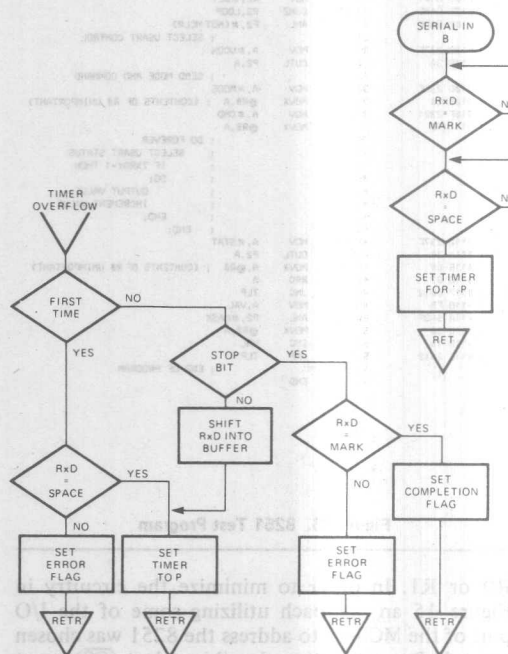


Figure 19. Improved Serial Input Routine

Using the timer to implement time delays as shown in Figure 19 results in considerable savings in processing time; two problems remain, however, which must be solved before an adequate software solution to the problem of receiving serial code can be found. The first problem is that even though the delays between bit samples are implemented via the timer rather than program loops the loop construction is still used to detect the leading edge of

the START bit. Although this results in the waste of processing power, the second problem is even more serious. For longer messages the required accuracy of the clocks becomes more and more stringent. Using the sampling technique discussed a cumulative error of one half a bit time in the time at which a bit sample is taken will result in erroneous reception. The maximum timing error which can be tolerated and yet still allow proper detection of an 11 bit ASCII character is then:

$$E_{max} = \frac{0.5 \cdot \text{BIT TIME}}{\text{CHARACTER TIME}} - \frac{0.5P}{11P} = 4.5\%$$

where P is the period of single bit. The corresponding calculation for a 32 bit character yields:

$$E_{max} = \frac{0.5P}{32P} = 1.6\%$$

Since this calculation does not allow for distortion on the signals, it is obvious that either extremely stable clocks will be required or a more tolerant algorithm must be devised. This problem is particularly serious at relatively high baud rates where the resolution of the counter (80μsecs with a 6 MHz crystal) becomes a significant percentage of the period of the received signal. At the 110 baud rate of the Teletype the 80μsec resolution of the clock allows a maximum accuracy of 0.33%; at 2400 baud this figure is reduced to 3.8%.

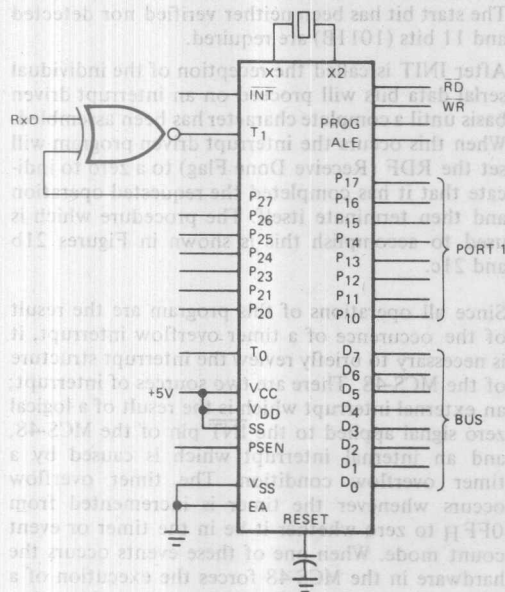


Figure 20. Detecting Rx/D Edges

Both efficient detection of the start bit and increased timing accuracy can be obtained if the MCS-48 can detect edges on the incoming received data (Rx/D). A hardware construct which allows this is shown in Figure 20.

The received data (Rx/D) is Exclusive NORed with bit seven of port two and fed into the TEST (T1) pin of the MCS-48. By manipulating P27 the program can now cause T1 to be either Rx/D or $\overline{\text{Rx/D}}$. (If P27 = 1 then T1 = Rx/D; if P27 = 0 then T1 = $\overline{\text{Rx/D}}$.) Note that not only can T1 be tested directly by the software but that it is the input which is used when the MCS-48 timer is in the event counter mode. The significance of this will be discussed later. The relationship between T1, P27, and Rx/D is given by the Boolean expression:

$$T1 = P27 \cdot \overline{\text{Rx/D}} + \overline{P27} \cdot \text{Rx/D}$$

Figure 21 flowcharts a means of utilizing this hardware construct to avoid the necessity of wasting time in program loops to detect the leading edge of the start bit. The receive operation is initialized when the program desiring to receive serial data calls the INIT subroutine (Figure 21a). Since INIT is going to manipulate the timer the first action it performs is to disable the timer overflow interrupt. Its next step is to set P27 to a logical 1. Setting P27 in this manner causes the TEST 1 input to the MCS-48 to follow $\overline{\text{Rx/D}}$. By setting up the receive circuitry in this manner a high to low transition will occur on TEST 1 when the Rx/D goes from the MARKING to SPACING state (i.e. the START

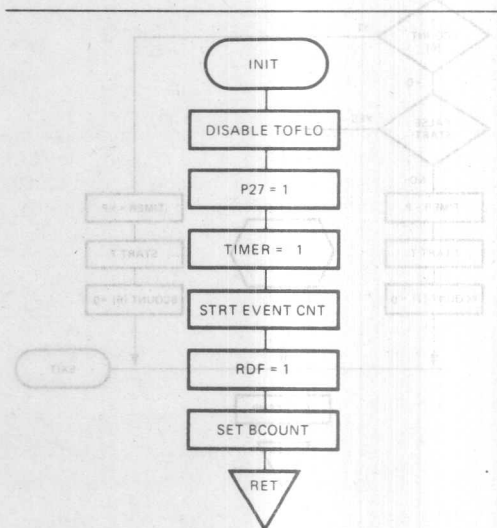
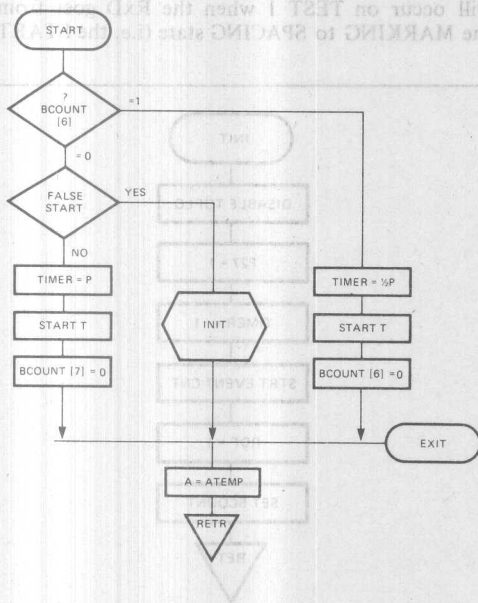
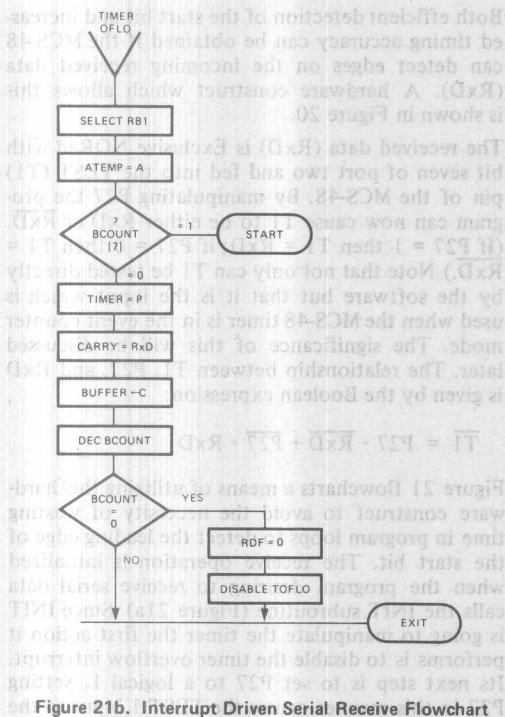
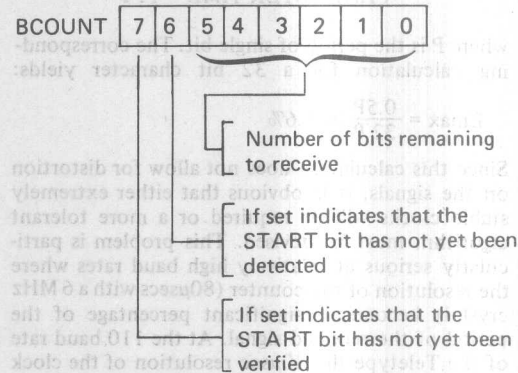


Figure 21a. Interrupt Driven Serial Receive Flowchart



bit occurs). By setting the timer to OFFH and enabling it in the event count mode, the INIT routine sets up the MCS-48 to generate a timer overflow interrupt on the next MARK to SPACE transition of RxD (the TEST 1 input doubles as the event counter input). Before returning to the calling program the INIT routine sets a flag (RDF) which will be cleared by the receive program when the requested receive operation is complete. INIT also sets a value into a register called BCOUNT. The receive program interprets BCOUNT as follows:



In order to request the reception of the 11 bit ASCII code INIT would set BCOUNT to 11001011B. The start bit has been neither verified nor detected and 11 bits (1011B) are required.

After INIT is called the reception of the individual serial data bits will proceed on an interrupt driven basis until a complete character has been assembled. When this occurs the interrupt driven program will set the RDF (Receive Done Flag) to a zero to indicate that it has completed the requested operation and then terminate itself. The procedure which is used to accomplish this is shown in Figures 21b and 21c.

Since all operations of this program are the result of the occurrence of a timer overflow interrupt, it is necessary to briefly review the interrupt structure of the MCS-48. There are two sources of interrupt; an external interrupt which is the result of a logical zero signal applied to the INT pin of the MCS-48, and an internal interrupt which is caused by a timer overflow condition. The timer overflow occurs whenever the timer is incremented from OFFH to zero whether it be in the timer or event count mode. When one of these events occurs the hardware in the MCS-48 forces the execution of a CALL. This CALL has a preset address of location 3 if it is due to the external interrupt and location 7 if it is due to a timer overflow. If both of these

events occur simultaneously the external interrupt will take precedence. The CALL automatically saves the contents of the program counter for the running program and its PSW (program status word) on a stack the hardware maintains in RAM locations 8-23. Although the hardware saves the program counter and PSW, it remains the responsibility of any interrupt driven software to make absolutely certain that it does not modify any memory locations or registers which are being used by the main program. The most convenient way of ensuring this in the MCS-48 is to dedicate the second bank of registers (RB1) to the interrupt driven program. One of these registers has to be used to save the accumulator (which is not part of the register bank) but seven registers remain; including two which can be used as pointers to the rest of the RAM (R0 and R1). Note that if this approach is taken then these registers have to be allocated between the program which services the external interrupt and the one which services the timer overflow. This problem is somewhat alleviated by a hardware lockout which prevents the timer overflow interrupt from interrupting the external interrupt service routine and vice versa. This is implemented by locking out new interrupts between the time an interrupt is recognized and the time a RETR instruction is executed. The RETR instruction is like a normal RET (return from subroutine) except that the PSW as well as the program counter is restored. The RETR instruction can be very much thought of as a return from interrupt instruction in the MCS-48.

The receive program under discussion uses register bank 1 in the manner described. Whenever a timer overflow occurs (e.g. on the next MARK to SPACE transition of Rx/D after INIT is called), control is passed (by the hardware generated CALL) to the point labeled TIMER OFLO in Figure 21b. This program segment immediately selects register bank 1 (RB1) and then saves the accumulator (A) in a location called ATEMP which is actually R7 of RB1. The program then tests bit seven of BCOUNT (R6 of RB1) to find out if a START bit has been verified (i.e. the edge of the START bit has first been detected and then verified to still be a SPACE one-half a bit time later. If BCOUNT [7] is a zero the START has been verified and the program proceeds to set the timer to P (the period of the serial bit), get the current serial data into the carry bit, and then shift the carry bit into a buffer. After saving the data the program decrements BCOUNT and tests it for zero. If BCOUNT is zero the receive operation is complete so the program sets RDF to a zero and disables timer overflow interrupts. Whether or not BCOUNT is zero, control is passed to EXIT where A is loaded with ATEMP and a

RETR is executed. Note that since the state of the flip flop which selects RB1 is saved as part of the PSW, the execution of RETR automatically selects the register bank which was active when the interrupt occurred.

If BCOUNT [7] is still set when it is tested, control is passed to START (Figure 21c) where bit 6 is tested to determine if the START has been detected yet. If BCOUNT [6] is set it indicates that this is the first occurrence of a timer overflow since the receive process was initialized by the INIT subroutine. If this is so, the program assumes that the START bit has just started and therefore it sets the timer to one-half of a bit time ($1/2 P$), starts the timer in the timer mode, and clears BCOUNT [6] to indicate that the START bit has been detected. The next overflow will again result in the execution of the program in Figure 21b and again BCOUNT [7] will be found to be set. This time, however, BCOUNT [6] will be reset and the program will know that it should test the START bit to ensure that it is still a SPACE. This test is performed and if successful the timer is set for a bit period P and BCOUNT [7] is reset so that on the next occurrence of a timer overflow the program will know that it should start assembling serial bits into a character. If the test is unsuccessful, the subroutine INIT is used to reinitialize the receive program. In either case control is passed to EXIT where a return from interrupt mode occurs.

This receive program, listings of which appear in Figure 22, allows the reception of serial characters transparently to the main running software. After INIT is called the main program has only to check RDF periodically to find out if there is data in the buffer for it. It would be fairly easy to 'double buffer' this operation by providing a buffer which the receive program uses to deserialize the incoming code and a second buffer to store the assembled character. If the program would reinitialize itself upon completion, the reception of a string of characters could proceed in much the same way as it would if a status driven USART were being used.

Although this program solves the first problem of software controlled reception (lack of efficiency) the second problem—sensitivity to frequency variations—remains. An example of a code which would be susceptible to this problem is the 31,26 BCH code commonly used in supervisory control systems. (A supervisory control system is, in essence, a remote control system which allows a human or computer operator the control of a system via a serial communications link.) The BCH codes are used because of their error detection capabilities and are a class of cyclical redundancy

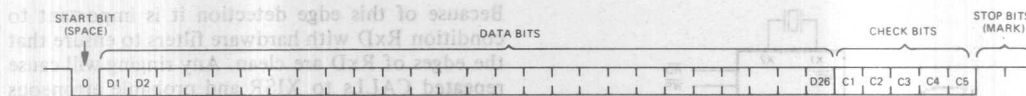


Figure 23. 31,26 BCH Code

A concept which reduces sensitivity to frequency deviations and thus allows the reception of longer codes is shown pictorially in Figure 24. The first line of this timing chart shows an alternative ones and zeros pattern on the RxD with a period of 5 milliseconds. The second line shows that by sampling at a period of exactly 5 milliseconds the data can be properly interpreted. The third and fourth lines show the effects of sampling with a period of six and four milliseconds respectively. In either case, an error occurs at the third sample where both periods result in sampling on an edge of the RxD signal. The third line of Figure 24 shows a hybrid sampling scheme which, based on some additional information, switches sampling periods between the two values. As can be seen in Figure 24, the data is sampled with a 4 millisecond period until the sampling begins to fall behind the data; at this point the sampling period is increased to six milliseconds and the sampling first catches up and then passes the center point of the data. As soon as this happens, the sampling period reverts to the 4 millisecond period and the cycle repeats. It can be seen that this scheme sets up a pattern which repeats indefinitely and the data can be successfully sampled. Note that the sampling pattern established is alternating periods of four and six milliseconds. The average period of this pattern, as might be expected, is 5 msec. Line 5 of Figure 24 shows the effect of a change in transmission speed to a period of 5.5 msec with no change in the sampling time. The sampling is again successful but the new sampling pattern is 4-6-6-6; 4-6-6-6, etc. Note that the average sample is again equal to the period of the received data (5.5). While this scheme

does seem to work, the question of what additional information is needed remains.

The MCS-48 must somehow decide when it is drifting out of synchronization and take corrective action. By referring back to Figure 24 it can be seen that if the MCS-48 could determine where the edges of RxD occurred with respect to its sampling times then the additional information would be available. As can be seen in the figure the choice of sampling period can be based on the following rule:

If an edge on the RxD line occurs during the first half of the current sampling period, then use the short period for the next sample. If an edge occurs during the second half of the period, then use the long sampling period for the next sample.

If the data on the RxD line does not change, of course, the MCS-48 will drift out of synchronization just as the original algorithm did. As long as edges occur on TxD, however, synchronization can be maintained. To maximize the allowable time between edges, the following addition could be made to the above rule:

If no edge occurs on the RxD line during a sample, then change sampling period from short to long or vice versa.

Note that this addition to the rule will result in using an average of the two sampling periods when no edge occurs for several bit times.

The edges of RxD can be easily detected by the use of the same structure (the Exclusive — NOR gate) which was added to the MCS-48 in Figure 20. This gate, which is used to detect the edge on RxD which begins the START bit, can naturally be used to detect any edge. Since the timer is being used to time the bit period, however, the event count input (T1) is not useful during the receive itself. By connecting the output of this gate, however, to the INT input to the MCS-48 (see Figure 25) it is possible to detect edges on RxD with the event counter when the program is trying to detect the START bit and by the external interrupt when the program is using the timer to control the sampling times.

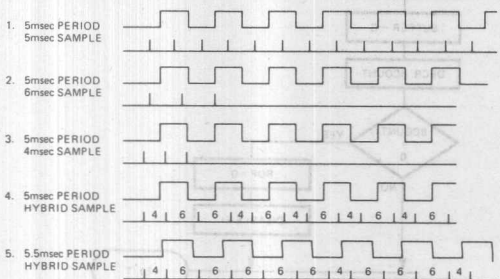


Figure 24. Various Sampling Alternatives

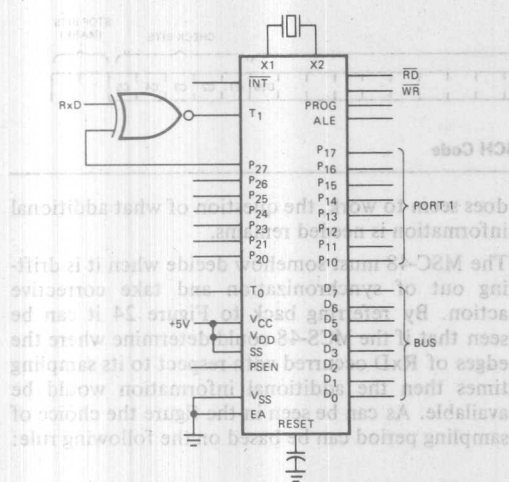
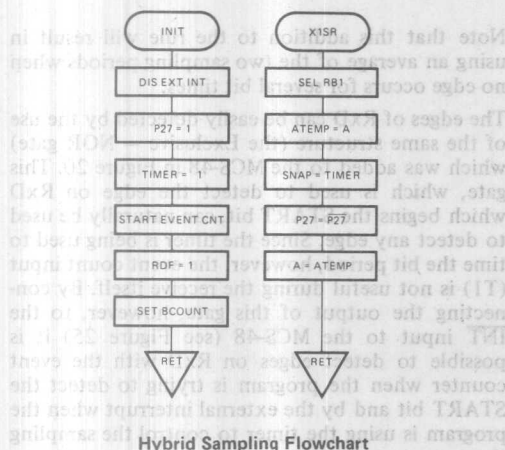


Figure 25. Modified Edge Detection

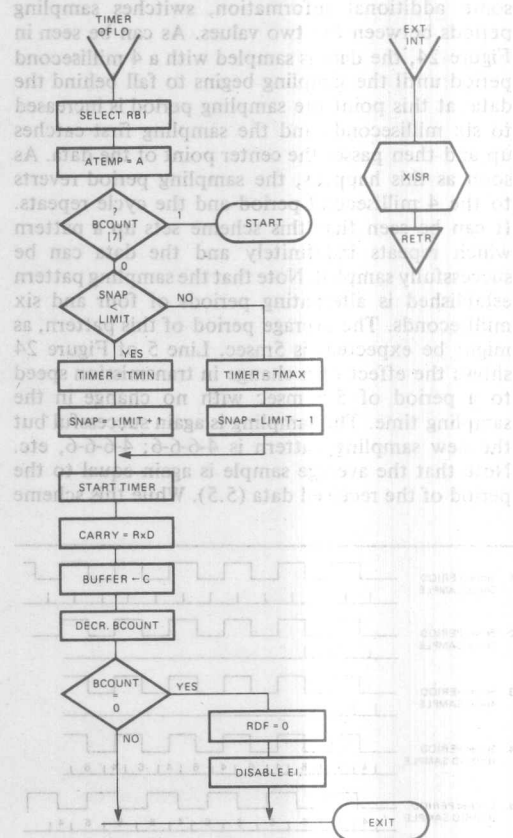
A modification to the program of Figure 21 which implements this new sampling algorithm is shown in Figure 26. The first deviation from the original program is the addition of a routine (XISR, Figure 26a which is called when an external interrupt occurs (i.e. when an edge occurs on RxD). This routine saves the status of the running program and then stores the current value of the timer register in a location called SNAP (R5 of RB1). After doing these operations the program complements bit 7 of port 2. Manipulating P27 in this manner will cause the Exclusive NOR gate to turn off the external interrupt and will set it up to generate another interrupt when the RxD line changes again (has another edge).



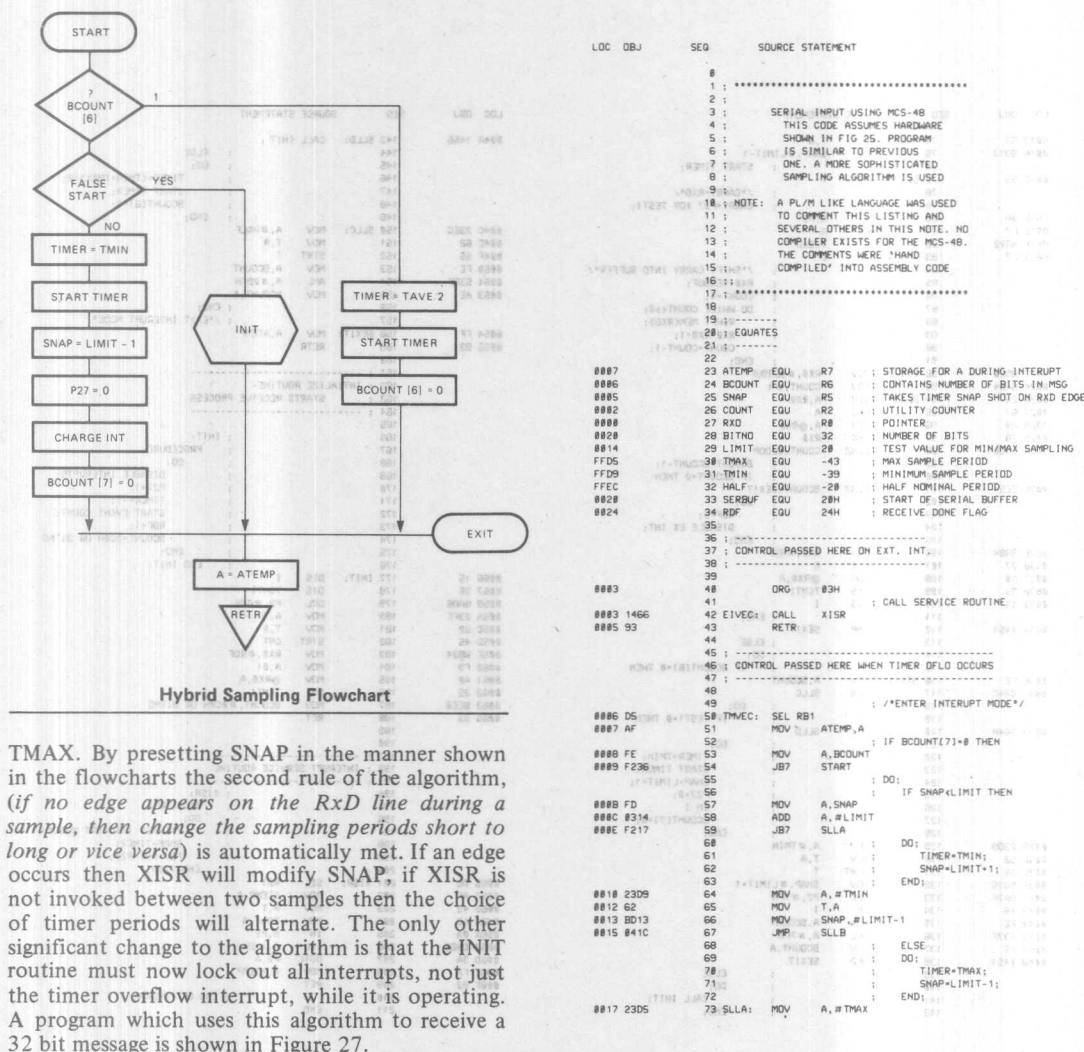
Hybrid Sampling Flowchart

Because of this edge detection it is important to condition RxD with hardware filters to ensure that the edges of RxD are clean. Any ringing will cause repeated CALLs to XISR and probable erroneous operation. The changes to the START process (Figure 26c) are two-fold; first the TIMER is set to one half the average of the two sample periods when the START bit is first detected (BCOUNT [6] = 1), and second the processing of the edge information is initialized by presetting SNAP and clearing P27.

SNAP is preset so that when the reception of data actually begins (Figure 26b BCOUNT [7] = 0), the decision block which tests SNAP against LIMIT will be initialized. This block actually compares the value in SNAP with a LIMIT value which is used to determine if the sampling point is ahead or behind the actual midpoint of the serial data. If the sampling is ahead then the timer is set for TMIN; if the sampling is behind then the timer is set for



Hybrid Sampling Flowchart



```

LOC OBJ   SEQ 44-234 SOURCE STATEMENT
0019 62    74 MOV T,A
001A BD13  75 MOV SNAP,#LIMIT-1
001C 55    76 START TIMER;
001D 8A    77 SLLB: STRT T
001E F7    78 /*CARRY-RXD*/
001F 4622  79 CARRY-P27 XOR TEST;
0021 A7    80 IN T,A,P2
0022 B828  81 RLC A,P2
0023 28    82 JNT1 TISR
0024 8A    83 CPL C
0025 85    84 /*SHIFT CARRY INTO BUFFER*/
0026 86    85 COUNT+4;
0027 67    86 DO WHILE COUNT<#;
0028 28    87 /*SHIFT MEMRXD;
0029 18    88 RLB-RXB+1;
002A EA26  89 COUNT-COUNT-1;
002B 8A    90 END;
002C B828  91 TISR: MOV RXB,#SERBUF
002D BA4  92 MOV COUNT,#4
002E 28    93 SLOOP: XCH A,RXB
002F 67    94 RRC A
0030 28    95 XCH A,@RXB
0031 18    96 INC RXB
0032 EA26  97 DJNZ COUNT,SLOOP
0033 8A    98 BCOUNT=BCOUNT-1;
0034 CE54  99 DJNZ BCOUNT,SEXIT
0035 8A    100 /*RD#-#;
0036 8A    101 RD#-#;
0037 8A    102 /*DISABLE EX INT;
0038 8A    103 END;
0039 27    104 MOV RXB,#RD#
003A A8    105 CLR A
003B 35    106 MOV @RXB,A
003C 35    107 DIS TCHT1
003D 15    108 DIS I
003E 454  109 JMP EXIT
003F FE    110 START: MOV A,BCOUNT
0040 D24C  111 JBE SLLC
0041 564A  112 IF TEST1=0 THEN
0042 564A  113 DO;
0043 564A  114 IF BCOUNT(G)=0 THEN
0044 564A  115 DO;
0045 564A  116 TIMER=THIN;
0046 564A  117 START TIMER;
0047 564A  118 SNAP=INIT+1;
0048 564A  119 P27=0;
0049 564A  120 EN I
0050 564A  121 BCOUNT(7)=0;
0051 564A  122 END;
0052 564A  123 DO;
0053 564A  124 MOV A,#THIN
0054 564A  125 MOV T,A
0055 564A  126 STRT T
0056 564A  127 MOV SNAP,#LIMIT-1
0057 564A  128 ANL P2,#7FH
0058 564A  129 EN I
0059 564A  130 MOV A,BCOUNT
0060 564A  131 ANL A,#7FH
0061 564A  132 MOV BCOUNT,A
0062 564A  133 JMP EXIT
0063 564A  134 ELSE
0064 564A  135 DO;
0065 564A  136 CALL INIT;
0066 564A  137 END;

```

```

LOC OBJ   SEQ 143-156 SOURCE STATEMENT
004A 1456 143 SLLD: CALL INIT
004B 1456 144 ELSE
004C 1456 145 DO;
004D 1456 146 TIMER=(THIN+THAX)/2;
004E 1456 147 START TIMER;
004F 1456 148 BCOUNT(G)=0;
0050 1456 149 END;
0051 23EC 150 SLLC: MOV A,#HALF
0052 62    151 MOV T,A
0053 55    152 STRT T
0054 FE    153 MOV A,BCOUNT
0055 53BF  154 ANL A,#BFH
0056 AE    155 MOV BCOUNT,A
0057 55    156 END;
0058 FF    157 SEXIT: MOV A,ATEMP
0059 93    158 RETR
0060 1456 159 INITIALIZE ROUTINE;
0061 1456 160 STARTS RECEIVE PROCESS
0062 1456 161 INIT:
0063 1456 162 DO;
0064 1456 163 DISABLE INTERRUPTS;
0065 1456 164 P27=1;
0066 1456 165 TIMER=1;
0067 1456 166 START EVENT COUNT;
0068 1456 167 RD#-1;
0069 1456 168 BCOUNT=BCOH OR BITNO
0070 1456 169 END;
0071 1456 170 END INIT;
0072 1456 171 INIT: DIS I
0073 1456 172 TCHT1
0074 1456 173 P2,#BFH
0075 1456 174 MOV A,#-1
0076 1456 175 MOV T,A
0077 1456 176 STRT CNT
0078 1456 177 MOV RXB,#RD#
0079 1456 178 MOV A,#1
0080 1456 179 MOV @RXB,A
0081 1456 180 EN TCHT1
0082 1456 181 BCOUNT,#COH OR BITNO
0083 1456 182 RET
0084 1456 183
0085 1456 184
0086 1456 185
0087 1456 186
0088 1456 187
0089 1456 188
0090 1456 189
0091 1456 190
0092 1456 191
0093 1456 192
0094 1456 193
0095 1456 194
0096 1456 195
0097 1456 196
0098 1456 197
0099 1456 198
0100 1456 199
0101 1456 200
0102 1456 201
0103 1456 202
0104 1456 203
0105 1456 204
0106 1456 205
0107 1456 206
0108 1456 207
0109 1456 208
0110 1456 209
0111 1456 210
0112 1456 211

```

This program is designed to initialize the hardware and software for the hybrid sampling program. It starts by calling the INIT routine, which sets up the timer, event counter, and interrupt mode. The main loop then enters a receive process, where it waits for a valid sample. When a sample is received, the program checks if it is a valid sample (P27=1) and if the timer is valid (TIMER=1). If both conditions are met, the program starts the event counter and sets the RD# to 1. The program then enters a loop where it checks if the BCOUNT is equal to the BCOH or BITNO. If it is, the program ends. Otherwise, it continues the loop.

Figure 27. Hybrid Sampling Program

Figure 27. Hybrid Sampling Program

Serial transmission is conceptually far simpler than serial reception since no synchronization is required. All that is required is to use the timer to generate interrupts at the bit rate and present the character to be transmitted serially at an I/O pin. A program which does this is shown in Figure 28. The transmission of serial data becomes much more complicated if it must occur simultaneously with reception.

If both reception and transmission are to occur simultaneously then obviously contention will exist for the use of the timer. It is possible to allow the simultaneous reception and transmission of serial data using the timer as a general clock which controls software maintained timers. The attainable baud rates using such techniques are, however, limited and the use of a 8251 USART is probably

indicated in all but the most cost sensitive applications. An exception to this rule occurs when the system, although full duplex in nature, actually transmits the same data as it receives. An example of this is a microprocessor driving a terminal such as a Teletype. Although the circuit to the terminal is full duplex, the data that is transmitted is generally the same as that received. A minor modification to the program shown in Figure 26 would implement this mode of operation. The modification would be to the XISR routine and it would add the code necessary to place the TxD I/O pin in the same state as the RxD line. Since any change in RxD results in a call to XISR, this modification would cause the retransmission of any received data. Whenever it becomes necessary to transmit data which is not being received, the program of Figure 28 could be used in a half duplex manner.

```

LOC  OBJ      SEQ      SOURCE STATEMENT
      0
1 : -----
2 : SERIAL TRANSMIT ON THE MCS48
3 : TO USE PUT A CHAR IN BUFF AND
4 : SET CHARAV TO 0FFH, WHEN THE
5 : TRANSMITTER IS READY FOR ANOTHER
6 : CHAR IT WILL CLEAR CHARAV. THE
7 : TRANSMISSION IS DOUBLE BUFFERED.
8 : -----
9
10 : -----
11 : EQUATES
12 : -----
13
0007 14 ATMP EQU R7      STORAGE FOR A DURING INT.
0008 15 PTOS EQU R6      PARALLEL TO SERIAL CONVERTER
0009 16 BUFF EQU R5      CHARACTER BUFFER
000A 17 CHARAV EQU R4     CHARACTER AVAILABLE FLAG
000B 18 COUNT EQU R3     BIT COUNTER
000C 19 CBIT EQU 0FFH    MASK TO CLEAR TXD IN P24
000D 20 SBIT EQU 010H    MASK TO SET TXD IN P24
000E 21 P EQU -41        PERIOD OF TXD
000F 22
0010 23 : -----
0011 24 : CONTROL PASSED HERE ON TIMER OVERFLOW
0012 25 : -----
0013 26 ORG 07H          ENTER INTERRUPT MODE
0014 27
0015 28 T0FLO: SEL RB1    SET TIMER FOR P
0016 29 MOV ATMP,A
0017 30
0018 31 MOV A,#P
0019 32 MOV T,A
001A 33 STR T
001B 34
001C 35 CALL BIT        GET BIT INTO CARRY
001D 36                SET TXD TO CARRY

```

```

LOC  OBJ      SEQ      SOURCE STATEMENT
      0
000F 37 IN A,P2
0010 38 XRL A,#00H
0011 39 OUTL P2,A
0012 40 JC BITON
0013 41 RNL P2,#CBIT
0014 42 JMP EXIT
0015 43 BITON: DRL P2,#SBIT
0016 44 EXIT: MOV A,ATMP
0017 45 RETR
0018 46
0019 47 : -----
0020 48 : BIT ROUTINE
0021 49 : -PICKS THE NEXT BIT TO TRANSMIT
0022 50 : -----
0023 51
0024 52 BIT: MOV A,COUNT
0025 53 JZ IDLE
0026 54 MOV A,PTOS
0027 55 RRC A
0028 56 DRL A,#00H
0029 57 MOV PTOS,A
0030 58 DEC COUNT
0031 59 RET
0032 60
0033 61 IDLE: CLR C
0034 62 MOV A,CHARAV
0035 63 JNZ GOTONE
0036 64 CPL C
0037 65 RET
0038 66
0039 67 GOTONE: MOV A,BUFF
0040 68 MOV PTOS,A
0041 69 MOV COUNT,#10
0042 70 MOV CHARAV,#0
0043 71 RET
0044 72 : END OF PROGRAM
0045 73 END

```

Figure 28. Serial Transmission

GENERATING PARITY

Many communications schemes require the generation and checking of parity. If a USART is used it can be programmed to automatically generate and check parity. If the communications is handled by software within the MCS-48™ then the program must perform parity calculations. Calculating parity is easy if one remembers what parity really means. A character has even parity if the number of one bits in it is even. A character has odd parity if it has an odd number of ones. The program segment shown in Figure 29 can be caused to calculate parity. It starts by setting a loop count to eight and

```

LOC  OBJ  SEQ  SOURCE STATEMENT
0
1
2 *****
3
4 : PARITY
5 : THIS PROGRAM GENERATES PARITY
6 : ON THE ACCUMULATOR
7 : CARRY WILL BE SET IF A HAS ODD PARITY
8
9 *****
10
11
12 :-----
13 : EQUATES
14 :-----
15
16 COUNT EQU R2
17
18
19 PAR: ORG 100H
20 COUNT, #8
21 : SET LOOP COUNT
22 : INITIALIZE CARRY
23 : FOR EACH ZERO BIT IN A
24 : COMPLEMENT THE CARRY FLAG
25
26 LOOP: RR A
27 : OVER
28 : CARRY
29 : END OF PROGRAM
30
31 END

```

Figure 29. Parity Generation

clearing the CARRY flag. After this initialization a loop is executed eight times. During each execution the accumulator is rotated and the least significant bit is tested. If the bit is a zero the CARRY flag is complemented, if the bit is a one no further action is taken. Since an even number of zeros implies an even number of ones for an eight bit character, after all eight loops have been accomplished the CARRY bit will be set if an odd number of ones were encountered; it will be reset if the number were even. Since the RR instruction does not involve CARRY the net result of executing this program loop is to set CARRY if parity is odd without effecting the character in the accumulator.

CONCLUSION

This Application Note has presented a very small sampling of the application techniques possible with the MCS-48™ family. The application of this new single chip computer system to tasks which have not yet yielded to the power of the microprocessor will present a fascinating challenge to the system designer.

INTRODUCTION

This application note presents a software package for interfacing members of Intel's MCS-48™ family of single-chip microcomputers with keyboards and displays using a minimum of external components. Because of the similarity of the architectures of the various members of the family (the 8035, 8048, 8748, 8039, 8049, 8021, and 8022 microcomputers; also the 8041 and 8741 universal peripheral interfaces in the UPI-41® family), the code included here could run with minor modifications on any member of the family.

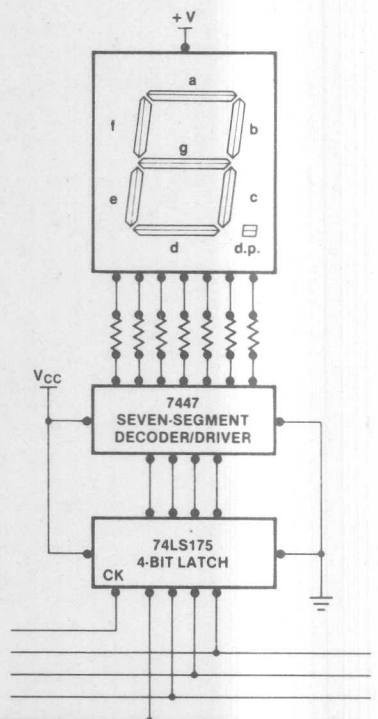
Since keyboard and display logic can be just one of several functions handled by a microprocessor, the added cost of including these functions in a system is minimal. In fact, considering the extremely low cost of standard X-Y matrix keyboards and integrated displays, their use is often more cost effective than even a handful of discrete switches and indicators. Thus, the additional flexibility of keyboard input and display output can be added to inexpensive consumer products (such as games, clocks, thermostats, tape recorders, etc.), while producing a net savings in system cost.

Since each potential application will have its own unique combination of keys and display characters, the program is written so that very little modification is needed to interface it with a wide variety of hardware configurations. In general, the only changes required are within the set of initial EQUates at the beginning of the program.

Along with the basic software for driving a multiplexed display and/or scanning and debouncing an X-Y matrix of key switches, a collection of utility subroutines is also included for implementing the most commonly used keyboard and display utility functions, such as copying simple messages onto the display or determining the encoded value of each key in the key matrix. As a result of the versatile architecture and applications-oriented instruction set of the MCS-48 family, the entire package fits into about 250 bytes of internal program ROM or EPROM, leaving the rest of the ROM space for the program to cook the perfect piece of toast, or whatever. By tailoring the software to match a known hardware configuration, or by selecting only those functions needed for a given application, the program size could be even further reduced.

Since what is being presented in this application note is a software package, rather than the usual hardware/software system design, the format of this note is somewhat different from most — it consists primarily of a long program listing reproduced in the following pages. For the most part, the listing is self-explanatory, with comments introducing each subroutine and major code segment. Some parts of this introduction are reproduced in the program listing itself, explaining the configuration of the prototype system. However, an additional bit of explanation would make the listing easier to understand, especially for those readers unfamiliar with the concept of multiplexed displays and keyboards.

In traditional digital system design, various hardware registers or counters were used to hold binary or BCD values which had to be conveyed to the user. The standard way of presenting this information was by connecting each register to a seven-segment encoder (such as the 7447) driving a single display character, as represented by Figure 1. Thus, two ICs, seven current limiting resistors, and about 45 solder joints were required for each digit of output. Consider how traditional techniques might be (mis-)applied in designing a microprocessor system: the designer could add a latch, encoder, and resistors for each digit of the display. Still another latch and decoder could be used to turn on one of the decimal points (if used). The characters displayed could only be a sequence of decimal digits. In the same vein, a large matrix of key switches could be read by installing an MSI TTL priority encoder read by an additional input port. Not only would all this use a lot of extra I/O ports and increase the system price and part count drastically, but the flexibility and reliability of the system would be greatly reduced.



CIRCUIT REPEATED FOR EVERY DIGIT OF DISPLAY
(DOTS USED TO INDICATE SOLDER JOINTS)

Figure 1. Wrong Way to Design Multiple Digit Displays for Microcomputer Systems

be used to decrease costs, part count, and interconnections, while allowing a wider range of character types to be used on the display. The techniques used here are fairly typical of today's integrated subsystems designed especially for controlling keyboards and displays (such as in calculators or the Intel® 4269, 8278, and 8279 Keyboard/Display Controller Devices).

In a multiplexed display, all the segments of all the characters are interconnected in a regular two-dimensional array. One terminal of each segment is in common with the other segments of the same character; the other terminal is connected with the same segments of the other characters. This is represented schematically in Figure 2. A digit driver or segment driver is needed for each of these common lines.

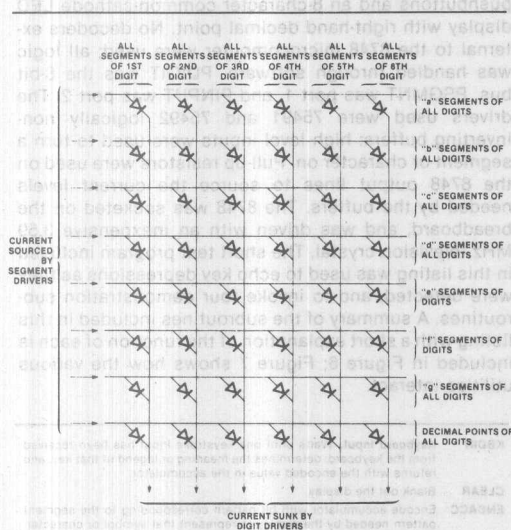


Figure 2. Schematic Representation of 6-Digit, 7-Segment Common-Cathode LED Multiplexed Display

The various characters of the display are not all on at once; rather, only one character at a time is energized. As each character is enabled, some combination of segment drivers is turned on, with the result that a digit appears on the enabled character. (For example, in Figure 3, if segment drivers 'a', 'b', and 'c' were on when character position #6 was enabled, the digit '7' would appear in the left-most place.) Each character is enabled in this way, in sequence, at a rate fast enough to ensure that the display characters seem to be on constantly, with no appearance of flashing or flickering.

In the system presented here, these rapid modifications to the display are all made under the control of the MCS-48™ microcomputer. At periodic intervals the computer quickly turns off all display segments, disables the character now being displayed and enables the next, looks up the pattern of segments for the next character

with the next character now turned on, the processor may now resume whatever it had been doing before. The whole display updating task consumes only a small fraction of the processor's time.

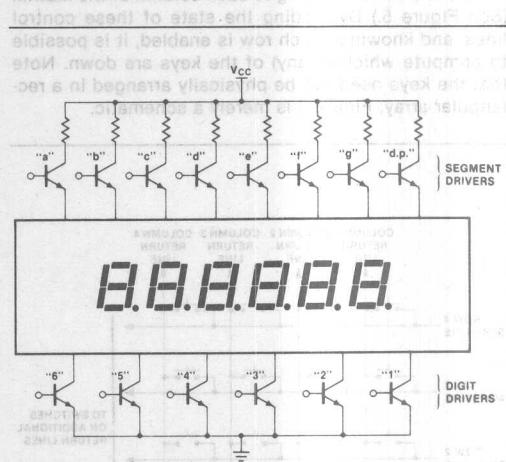


Figure 3. Segment and Digit Drivers used with 6-Position, 7-Segment LED Display

Moreover, since the computer rather than a standard decoder circuit is used to turn the segments off and on, patterns for characters other than decimal digits may be included in the display. Hexadecimal characters, special symbols, and many letters of the alphabet are possible. With sufficient imagination this feature can be exploited for some applications, as suggested by the examples in Figure 4.

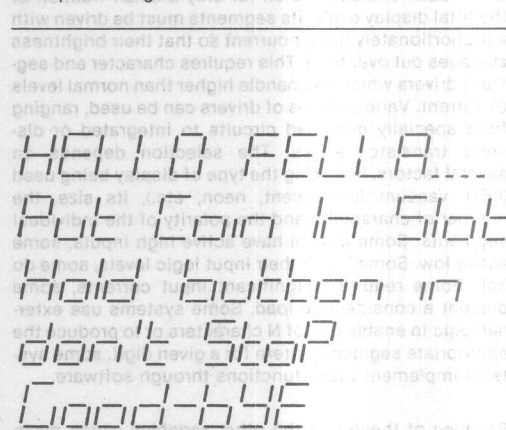


Figure 4. Examples of Typical Messages Possible with Simple 7-Segment Displays

As each character of the display is turned on, the same signal may be used to enable one row of the key matrix. Any keys in that row which are being pressed at the time will then pass the signal on to one of several "return lines", one corresponding to each column of the matrix. (See Figure 5.) By reading the state of these control lines, and knowing which row is enabled, it is possible to compute which (if any) of the keys are down. Note that the keys need not be physically arranged in a rectangular array; Figure 5 is merely a schematic.

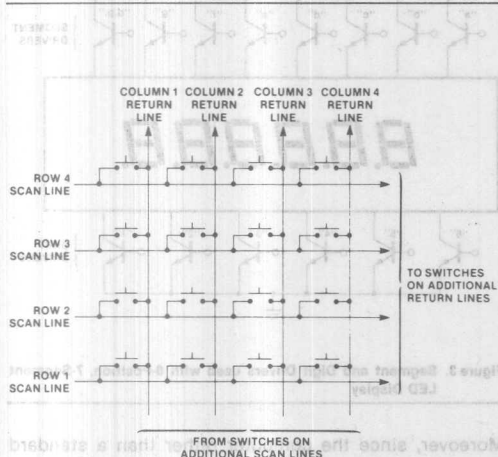


Figure 5. Schematic of X-Y Matrix Multiplexed Keyboard

Since each character is on for only a small fraction of the total display cycle, its segments must be driven with a proportionately higher current so that their brightness averages out over time. This requires character and segment drivers which can handle higher than normal levels of current. Various types of drivers can be used, ranging from specially designed circuits to integrated or discrete transistor arrays. The selection depends on several factors, including the type of display being used (LED, vacuum fluorescent, neon, etc.), its size, the number of characters, and the polarity of the individual segments. Some drivers have active high inputs, some active low. Some invert their input logic levels, some do not. Some require insignificant input currents, some present a considerable load. Some systems use external logic to enable one of N characters or to produce the appropriate segment pattern for a given digit, some systems implement these functions through software.

Because of these and the other variables which make each application unique, provisions are made in the first page of symbol EQUates to allow the user to specify such things as the number of characters in the display or the polarity of the drivers used, and the program will be assembled accordingly. The display is refreshed on each timer interrupt, which occurs every 32x (TICK)

machine cycles. (One machine cycle occurs every 30 crystal oscillations for the 8021 and 8022, or every 15 oscillations for all other members of the family.) A more detailed explanation of these variables is included in the listing.

Port assignment is also at the discretion of the user — all port references in the listing are "logical" rather than physical port names. The port used to specify which character is enabled is referred to as "PDIGIT". The output segment pattern is written to "PSGMNT" and the keyboard return lines are read by "PINPUT". These logical port names may be assigned to whichever ports the user pleases.

By way of example, the breadboard used to develop and debug this software used a matrix of 16 single-pole pushbuttons and an 8-character common-cathode LED display with right-hand decimal point. No decoders external to the 8748 microcomputer were used; all logic was handled through software. PDIGIT was the 8-bit bus, PSGMNT was port 1, and PINPUT was port 2. The drivers used were 75491 and 75492 logically non-inverting buffers: high level inputs were used to turn a segment or character on. Pull-up resistors were used on the 8748 output lines to source the current levels needed by the buffers. The 8748 was socketed on the breadboard, and was driven with an inexpensive 3.59 MHz television crystal. The short test program included in this listing was used to echo key depressions as they were detected, and to invoke four demonstration subroutines. A summary of the subroutines included in this listing with a short explanation of the function of each is included in Figure 6; Figure 7 shows how the various utilities interact.

KBDIN	Keyboard Input. Waits until one keystroke input has been received from the keyboard, determines the meaning or legend of that key, and returns with the encoded value in the accumulator.
CLEAR	Blank out the display.
ENCACC	Encode accumulator with bit pattern corresponding to the segment pattern needed by the display to represent that symbol or character. Uses the value of the accumulator when called to access a table containing the patterns for all legal input values.
WDISP	Write into Display. Writes the bit pattern in the accumulator into the next character position of the display. Maintains a character position counter so that repeated calls will automatically write characters into sequential positions.
RENTRY	Right-hand Entry. Stores the accumulator segment pattern in the display in the right-most character position. Shifts all other characters to the left one place.
PRINT	Print a string of arbitrary characters onto the display. Useful for prompting messages, warnings, etc. Uses a table of segment patterns in ROM, so that messages will not be restricted to numbers, letters, etc.
FILL	Fill the display with the character pattern in the accumulator. Useful for writing dashes, segment test patterns, etc., into all character positions.
ECHO	Wait for a key to be pressed by the operator and write that key onto the display. Used for providing feedback to the operator when entering numeric data, etc.
RDPADD	Adds or deletes a decimal point to the character at the right-hand side of the display, for entering floating point numbers.
HOLD	Called when a key is known to be down. Does not return until all keys have been released. Used for organ-type keyboards, or when some action should not be initiated until the key invoking that action has been released.
DELAY	Provides a crude real-time delay corresponding to the value of the accumulator when called. Can be used to cause display characters to blink, to momentarily flash information, to enable a buzzer, etc. Could also be used by the program when delays are needed, such as to slow down the computer reaction rate while playing a game against the human operator.

Figure 6. Utility Subroutine Definitions

LOC OBJ SEQ SOURCE STATEMENT

```

1 $MACROFILE XREF
2 $TITLE('AP40 INTEL MCS-48 KEYBOARD/DISPLAY APPLICATION NOTE APPENDIX')
3
4 THE FOLLOWING SOFTWARE PACKAGE PROVIDES A SEVEN SEGMENT DISPLAY
5 INTERFACE FOR MICROCOMPUTERS IN THE INTEL MCS-48 FAMILY
6 THE CODE IS WRITTEN SO THAT VARIOUS HARDWARE
7 CONFIGURATIONS CAN BE ACCOMMODATED BY REDEFINING THE INITIAL VARIABLES
8 IN MOST SITUATIONS, THE KEYBOARD/DISPLAY INTERFACE WILL BE REQUIRED TO
9 IMPLEMENT MORE SOPHISTICATED SINGLE-CHIP SYSTEMS (CALCULATORS, SCALES, CLOCKS,
10 ETC.), WITH SECTIONS OF THE FOLLOWING CODE SELECTED AND MODIFIED AS NECESSARY
11 FOR EACH APPLICATION
12
13 A SINGLE SUBROUTINE (CALLED REFRESH) IS USED TO IMPLEMENT BOTH THE DISPLAY
14 MULTIPLEXING AND KEYBOARD SCANNING, USING THE SAME SIGNAL BOTH TO ENABLE
15 ONE CHARACTER OF THE DISPLAY AND TO STROBE ONE ROW OF THE X-Y KEY MATRIX.
16 THE SUBROUTINE MUST BE CALLED SUFFICIENTLY OFTEN TO ENSURE THE DISPLAY
17 CHARACTERS DO NOT FLICKER- AT LEAST 50 COMPLETE DISPLAY SCANS PER SECOND
18 TO ACCOMMODATE SWITCHES OF ARBITRARY CHEAPNESS, THE DEBOUNCE TIME CAN BE
19 SET TO BE ANY DESIRED NUMBER OF COMPLETE SCANS
20 THUS THE DEBOUNCE TIME IS A FUNCTION OF BOTH THE SCAN RATE AND THE VALUE
21 OF CONSTANT 'DEBNC'
22
23 IN THIS LISTING, THE INTERNAL TIMER IS USED TO GENERATE INTERRUPTS THAT
24 SERVE AS A TIME BASE FOR THE REFRESH SUBROUTINE
25 ALTERNATE TIME BASES MIGHT BE AN EXTERNAL OSCILLATOR (DRIVING THE INTERRUPT
26 PIN OR POLLED BY A TEST OR INPUT PIN), A SOFTWARE DELAY LOOP IN THE BACKGROUND
27 PROGRAM, OR PERIODIC CALLS TO THE SUBROUTINE FROM THROUGHOUT THE USER'S PROGRAM
28 AT APPROPRIATE PLACES
29 IN THESE CASES, THE CODE STARTING AT LABEL TIINT (TIMER INTERRUPT) AND TIRET
30 (TIINT RETURN) COULD STILL BE USED TO SAVE AND RESTORE ACCUMULATOR CONTENTS
31 THE INTERRUPT SERVICING ROUTINE SELECTS REGISTER BANK 1
32 FOR THE NEEDED REGISTERS
33
34
35 WRITTEN BY JOHN WHARTON, INTEL SINGLE-CHIP COMPUTER APPLICATIONS
36
37 $EJECT
  
```

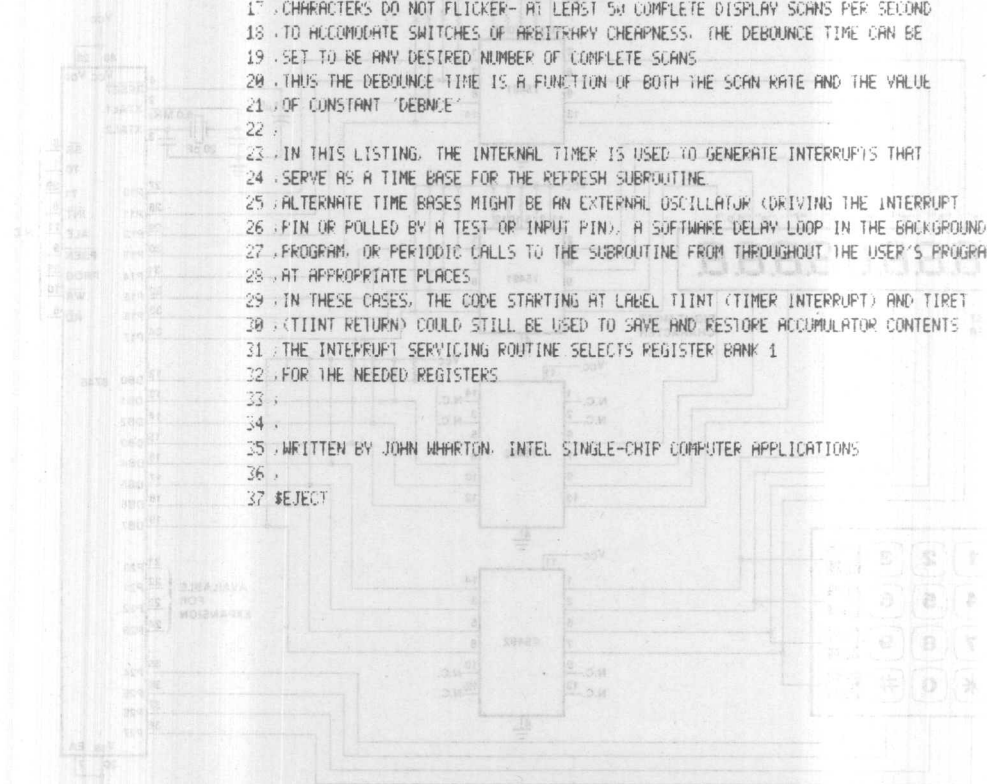


Figure 8 Keyboard/Display System Schematic

1515-11 MCS-48/UPI-41 MACRO ASSEMBLER, V2.0

AP40 INTEL MCS-48 KEYBOARD/DISPLAY APPLICATION NOTE APPENDIX

LOC	OBJ	SEQ	SOURCE STATEMENT
38			IN THIS IMPLEMENTATION OF THE DISPLAY SCAN, IT IS ASSUMED THAT THERE WILL
39			BE RELATIVELY LITTLE I/O OTHER THAN FOR THE KEYBOARD/DISPLAY
40			IF THIS IS THE CASE, THEN THERE IS NO NEED FOR FOR ANY ADDITIONAL EXTERNAL
41			LOGIC (SUCH AS ONE-OF-EIGHT DECODERS OR SEVEN-SEGMENT ENCODERS), THOUGH
42			THERE WILL STILL BE A NEED FOR CURRENT OR VOLTAGE DRIVERS, ACCORDING TO
43			THE TYPE OF DISPLAY BEING USED.
44			
45			IN THIS LISTING, THE PROCESSOR I/O PORTS ARE LOGICALLY DIVIDED AS FOLLOWS
46			
47			PDIGIT-EIGHT BIT PORT USED TO ENABLE, ONE AT A TIME, THE INDIVIDUAL
48			CHARACTERS OF AN EIGHT DIGIT SEVEN-SEGMENT DISPLAY, WHILE ALSO
49			STROBING THE ROWS OF AN X-Y MATRIX KEYBOARD.
50			BIT7 ENABLES THE LEFTMOST CHARACTER AND THE BOTTOM ROW OF THE KBD.
51			BIT4 ENABLES THE TOP ROW OF THE 4X4 KBD AND THE FOURTH CHARACTER.
52			BIT0 ENABLES THE RIGHTMOST CHARACTER.
53			(A 4X8 KEYBOARD COULD BE STROBED BY ALSO USING BIT3-BIT0
54			AND EXTENDING OR ELIMINATING THE TABLE "LEGND5".)
55			THE ENABLING OF ONE BIT (ACTIVE HIGH OR LOW) IS ACCOMMODATED BY
56			ACCESSING A LOOK-UP TABLE CALLED CHSTB.
57			THIS TECHNIQUE TAKES ABOUT FOUR BYTES MORE ROM THAN A TECHNIQUE
58			OF ROTATING A "ONE" THROUGH A FIELD OF "ZEROS" IN THE ACC
59			AN APPROPRIATE NUMBER OF TIMES, BUT IT ALLOWS SOME ADDITIONAL
60			FLEXIBILITY: IF THE DRIVERS BEING USED HAVE A COMBINATORIAL INPUT
61			(AS IN THE 7545X FAMILY OF HIGH-CURRENT, HIGH-VOLTAGE DRIVERS),
62			THE CHSTB TABLE COULD PROVIDE ENCODED OUTPUTS. NINE DIGITS, FOR
63			EXAMPLE, COULD BE ENABLED WITH SIX BITS OF (BUFFERED) OUTPUT
64			(001001, 001010, 001100, 010001, 010010, 010100, 100001, 100010, 100100)
65			IF I/O LINES NEED TO BE CONSERVED, OR IF MANY DIGITS
66			MUST BE DISPLAYED, AN EXTERNAL DECODER COULD BE ADDED TO THE SYSTEM
67			DURING CHARACTER TRANSITIONS A "BLANK" CHARACTER IS
68			EXPLICITLY WRITTEN TO THE DISPLAY. THUS,
69			THERE WILL BE NO CHARACTER "SHADOWING" CAUSED BY THE
70			FACT THAT THE HARDWARE OR SOFTWARE DECODER KEEPS ONE
71			OUTPUT, AND THUS ONE CHARACTER, ACTIVE AT ALL TIMES
72			
73			PSIGMNT-EIGHT BIT PORT TO ENABLE THE SEVEN SEGMENTS & D.P. OF A STANDARD
74			DISPLAY.
75			BIT7-BIT0 CORRESPOND TO THE DP AND SEGMENTS G THROUGH A, RESPECTIVELY.
76			IT IS POSSIBLE TO ACCOMMODATE
77			DRIVERS WHICH ARE EITHER LOGICALLY INVERTING OR NON-INVERTING BY
78			SETTING VARIABLE 'SEGPOL' (SEGMENT POLARITY).
79			NOTE THAT BY HAVING ARBITRARY CONTROL OVER EACH SEGMENT, NON-NUMERIC
80			CHARACTERS CAN BE REPRESENTED ON A SEVEN SEGMENT DISPLAY.
81			AS SHOWN IN EXAMPLE SUBROUTINE 'TEST2'.
82			
83			REJECT

ISIS-II MCS-48/UPT-41 MACRO ASSEMBLER, V2.0

AP40 INTEL MCS-48 KEYBOARD/DISPLAY APPLICATION NOTE APPENDIX I

LOC	OBJ	SEQ	SOURCE STATEMENT	THIRDSOURCE	LINE	OBJ
84			PINPUT-FOUR HIGH-ORDER BITS USED AT INPUTS FROM THE KEYBOARD RETURN LINES			
85			ASSUMES THAT A KEY DOWN IN THE CURRENTLY ENABLED ROW WOULD RETURN			
86			A LOW LEVEL			
87			IN THIS CASE, BIT7 RETURNS THE LEFTMOST COLUMN, BIT4 THE RIGHTMOST			
88			THE HIGH-ORDER BITS ARE USED SO THAT IF AN OFF-CHIP DECODER IS USED			
89			TO ENABLE UP TO 16 CHARACTERS, FOR EXAMPLE, IT COULD BE DRIVEN BY			
90			THE LOW ORDER BITS OF THE SAME PORT			
91			NOTE ALSO THAT IF A SIXTEEN KEY MATRIX WERE ELECTRICALLY ORGANIZED			
92			IN A 2X8 ARRAY, ONLY TWO RETURN LINES WOULD BE NEEDED			
93			(IN THIS CASE, PERHAPS T0 AND T1 COULD BE USED FOR INPUT BITS)			
94						
95			PULL-UP RESISTORS ON THE RETURN LINES MIGHT BE IN ORDER IF THERE IS ANY			
96			POSSIBILITY OF A HIGH-IMPEDENCE CONDUCTIVE PATH THROUGH THE SWITCH WHEN			
97			IT IS SUPPOSED TO BE OPEN			
98			(THIS PHENOMENON HAS ACTUALLY BEEN OBSERVED)			
99						
100			THE DRIVERS USED IN THE PROTOTYPE WERE ALL NON-INVERTING IN THAT			
101			A HIGH LEVEL ON AN OUTPUT LINE IS USED TO TURN A CHARACTER OR SEGMENT ON			
102			THERE ARE A TOTAL OF SEVEN I/O LINES LEFT OVER			
103						
104			THE ALGORITHM FOR DRIVING THE DISPLAY USES A BLOCK OF INTERNAL RAM			
105			AS DISPLAY REGISTERS, WITH ONE BYTE CORRESPONDING TO EACH CHARACTER OF THE			
106			DISPLAY. THE EIGHT BITS OF EACH BYTE CORRESPOND TO THE SEVEN SEGMENTS & DP			
107			OF EACH CHARACTER. IF AN EXTERNAL ENCODER IS USED (SUCH AS A FOUR-BIT TO			
108			SEVEN-SEGMENT ENCODER OR A ROM FOR TRANSLATING ASCII TO			
109			SIXTEEN-SEGMENT "STARBURST" DISPLAY PATTERNS), THE TABLE ENTRIES WOULD HOLD			
110			THE CHARACTER CODES. (IN THE FORMER CASE, AN UNUSED BIT COULD BE USED TO			
111			ENABLE THE D.P.)			
112			THUS, WRITING CHARACTERS TO THE DISPLAY FROM THE BACKGROUND PROGRAM			
113			REALLY ENTAILS WRITING THE APPROPRIATE SEGMENT			
114			PATTERNS TO A DISPLAY REGISTER. THE ACTUAL OUTPUTTING IS AUTOMATIC			
115			THE LEFTMOST CHARACTER CORRESPONDS TO THE LAST BYTE OF THE DISPLAY			
116			REGISTERS, AND IS ACCESED BY NEXTPL=8 (SEE SOURCE); THE RIGHTMOST			
117			CHARACTER IS THE FIRST DISPLAY BYTE, WHEN NEXTPL=1			
118			UTILITY SUBROUTINES ARE INCLUDED HERE TO TRANSLATE FOUR BIT NUMBERS TO HEX			
119			DIGIT PATTERNS, AND WRITE THEM INTO THE DISPLAY REGISTERS SEQUENTIALLY			
120			(EITHER FILLING FROM THE LEFT- H.P. CALCULATOR STYLE OR FROM THE			
121			RIGHT- T.I. STYLE. SUBROUTINES WDISP AND RENTRY, RESPECTIVELY,			
122						
123			THE KEYBOARD SCANNING ALGORITHM SHOWN HERE REQUIRES A KEY BE DOWN FOR			
124			SOME NUMBER OF COMPLETE DISPLAY SCANS TO BE ACKNOWLEDGED. SINCE IT IS			
125			INTENDED FOR 'ONE-FINGER' OPERATION, TWO-KEY ROLLOVER/N-KEY LOCKOUT HAS			
126			BEEN IMPLEMENTED. HOWEVER, MODIFICATIONS WOULD BE POSSIBLE TO ALLOW, FOR			
127			EXAMPLE, ONE KEY IN THE MATRIX TO BE USED AS A SHIFT KEY OR CONTROL KEY			
128			TO BE HELD DOWN WHILE ANOTHER KEY IN THE MATRIX IS PRESSED (SEE NOTE WITHIN			
129			THE BODY OF THE LISTING)			
130						
131			#EJECT			

LOC	OBJ	SR	SOURCE STATEMENT	SYMBOL	ADDRESS	HEX
132			BE AWARE THAT NO MORE THAN TWO KEYS CAN EVER BE DOWN UNLESS DIODES			
133			ARE PLACED IN SERIES WITH ALL OF THE SWITCHES- CERTAINLY NOT THE CASE FOR EL			
134			CHEAPO KEYBOARDS- BECAUSE SOME COMBINATIONS OF THREE KEYS DOWN WILL RESULT			
135			IN A 'PHANTOM' FOURTH KEY BEING PERCEIVED			
136			THE PHANTOM KEY WOULD BE THE FOURTH 'CORNER' WHEN THREE KEYS FORMING			
137			A RECTANGULAR PATTERN (IN THE X-Y KEY MATRIX) ARE DOWN X,Y			
138			IF DIODES ARE PLACED IN THE SCANNING ARRAY, CONSIDERATIONS MUST BE MADE			
139			ABOUT HOW THE DIODE VOLTAGE DROP WILL AFFECT INPUT LOGIC LEVELS			
140			PORT USED TO SCAN FOR KEY			
141			WHEN A DEBOUNCED KEY IS DETECTED, THE NUMBER OF ITS POSITION IN THE KEY			
142			MATRIX (LEFT-TO-RIGHT, BOTTOM-TO-TOP, STARTING FROM 00) IS PLACED INTO			
143			RAM LOCATION 'KBD0BUF' AN INPUT SUBROUTINE THEN NEED ONLY READ THIS LOCATION			
144			REPEATEDLY TO DETERMINE WHEN A KEY HAS BEEN PRESSED WHEN A KEY IS DETECTED,			
145			A SPECIAL CODE BYTE SHOULD BE WRITTEN BACK TO INTO 'KBD0BUF' TO PREVENT			
146			REPEATED DETECTIONS OF THE SAME KEY			
147			THE ROUTINE 'KBDIN' DEMONSTRATES A TYPICAL INPUT PROTOCOL ALONG WITH A METHOD			
148			FOR TRANSLATING A KEY POSITION TO ITS ASSOCIATED SIGNIFICANCE BY ACCESSING			
149			TABLE 'LEGND' IN ROM			
150			FOR TRILING CHARACTERS AND			
151			REJECT BITS USED AS			
152						
153			NUMBER OF DIGITS IN DISPLAY			
154			FORM OF KEYS (LESS THAN OR EQUAL TO CHARACT)			
155			LESSER DIMENSION OF KEYBOARD MATRIX			
156						
157			DETERMINE INTERVAL			
158			NUMBER OF SUCCESSIVE SCANS BEFORE KEY CLOSURE ACCEPTED			
159			CODE TO BLANK DISPLAY CHARACTERS			
160			WOULD BE SET TO ASCII DECODING FOR USED OR			
161			SEVEN-SEGMENT DECODER EXTERNAL TO			
162						
163			REFLECTS WHICH BITS ARE RELEVANT TO ENCODE SUBROUTINE			
164						
165						
166						
167						
168						
169						
170						
171						
172						
173						
174						
175						
176						
177						
178						
179						
180						
181						
182						
183						
184						
185						
186						
187						
188						
189						
190						
191						
192						
193						
194						
195						
196						
197						
198						
199						
200						
201						
202						
203						
204						
205						
206						
207						
208						
209						
210						
211						
212						
213						
214						
215						
216						
217						
218						
219						
220						
221						
222						
223						
224						
225						
226						
227						
228						
229						
230						
231						
232						
233						
234						
235						
236						
237						
238						
239						
240						
241						
242						
243						
244						
245						
246						
247						
248						
249						
250						
251						
252						
253						
254						
255						
256						
257						
258						
259						
260						
261						
262						
263						
264						
265						
266						
267						
268						
269						
270						
271						
272						
273						
274						
275						
276						
277						
278						
279						
280						
281						
282						
283						
284						
285						
286						
287						
288						
289						
290						
291						
292						
293						
294						
295						
296						
297						
298						
299						
300						
301						
302						
303						
304						
305						
306						
307						
308						
309						
310						
311						
312						
313						
314						
315						
316						
317						
318						
319						
320						
321						
322						
323						
324						
325						
326						
327						
328						
329						
330						
331						
332						
333						
334						
335						
336						
337						
338						
339						
340						
341						
342						
343						
344						
345						
346						
347						
348						
349						
350						
351						
352						
353						
354						
355						
356						
357						
358						
359						
360						
361						
362						
363						
364						
365						
366						
367						
368						
369						
370						
371						
372						
373						
374						
375						
376						
377						
378						
379						
380						
381						
382						
383						
384						
385						
386						
387						
388						
389						
390						
391						
392						
393						
394						
395						
396						
397						
398						
399						
400						
401						
402						
403						
404						
405						
406						
407						
408						
409						
410						
411						
412						
413						
414						
415						
416						
417						
418						
419						
420						
421						
422						
423						
424						
425						
426						
427						
428						
429						
430						
431						
432						
433						
434						
435						
436						

© 1978 Intel Corporation

LOC	OBJ	SEQ	SOURCE STATEMENT
		186	*****
		187	;
		188	BANK 0 REGISTERS USED
		189	;
		190	POINTERS USED FOR INDIRECT RAM ACCESSING:
0000		191	PNTR0 EQU R0
0001		192	PNTR1 EQU R1
0007		193	NEXTPL EQU R7 ;USED TO KEEP TRACK OF CHARACTER POSITION BEING
		194	;
		195	;
		196	*****
		197	;
		198	BANK 1 REGISTER ALLOCATION
		199	;
		200	PNTR0 EQU R0 (ALREADY DEFINED)
		201	PNTR1 EQU R1
0002		202	ASAVE EQU R2 ;HOLDS ACCUMULATOR VALUE DURING SERVICE ROUTINE
0004		203	ROTPAT EQU R4 ;USED TO HOLD INPUT PATTERN BEING ROTATED THROUGH CY
0005		204	ROTCNT EQU R5 ;COUNTS NUMBER OF BITS ROTATED THROUGH CY
0006		205	LASTKY EQU R6 ;HOLDS KEY POSITION OF LAST KEY DEPRESSION DETECTED
0007		206	CURDIG EQU R7 ;HOLDS POSITION OF NEXT CHARACTER TO BE DISPLAYED
		207	;
		208	*****
		209	;
		210	DATA RAM ALLOCATION
		211	;
0020		212	NREPTS EQU 32 ;KEEPS TRACK OF SUCCESSIVE READS OF SAME KEYSTROKE
0021		213	KEYLOC EQU 33 ;INCREMENTED AS SUCCESSIVE KEY LOCATIONS SCANNED
0022		214	KDBUF EQU 34 ;CARRIES POSITION OF DEBOUNCED KEY FROM REFRESH ROUTINE
		215	;\ BACK TO BACKGROUND PROGRAM
0023		216	RDELAY EQU 35 ;NON-ZERO WHEN DISPLAY IN PROGRESS
		217	;
		218	THE LAST <CHARNO> REGISTERS HOLD THE DISPLAY SEGMENT PATTERNS
		219	;
0037		220	SEGMAP EQU (63-CHARNO) ;BASE OF REGISTER ARRAY FOR DISPLAY PATTERNS
		221	;\ (COULD BE ANYWHERE IN INTERNAL RAM)
		222	;
		223	*****
		224	;
		225	NOTE THAT LASTKY, CURDIG, AND F1 RETAIN STATUS INFORMATION FROM
		226	ONE INTERRUPT TO THE NEXT. ALL OTHER REGISTERS MAY BE USED IN
		227	THE USER'S OWN INTERRUPT SERVICING ROUTINE
		228	;
		229	*****
		230	;
		231	\$EJECT

ISIS-II MCS-48/UPT-41 MACRO ASSEMBLER, V2.0

PAGE 17

AP40: INTEL MCS-48 KEYBOARD/DISPLAY APPLICATION NOTE APPENDIX A

LOC	OBJ	SEQ	SOURCE STATEMENT	TARGET STATEMENT	SEQ	LOC
		232	*****		100	
		233	*****		101	
		234	;		102	
0000		235	ORG 000H		103	
0000 0460		236	JMP 2223INIT		104	
		237	;		105	
		238	;		106	
		239	*****		107	
		240	;		108	
0007		241	ORG 007H		109	
		242	*****		110	
		243	TIINT TIMER INTERRUPT SUBROUTINE		111	
		244	CALL MADE TO LOC 007H WHEN TIMER TIMES OUT		112	
		245	TIMER CAN BE RE-INITIALIZED AT THIS POINT IF DESIRED.		113	
		246	USED HERE TO CAUSE THE DISPLAY REFRESH AND KEY SCAN ROUTINES TO		114	
		247	BE CALLED PERIODICALLY.		115	
0007 D5		248	TIINT SEL RB1		116	
0008 AA		249	MOV ASAVE, A		117	
0009 23F0		250	MOV A, #TICK		118	
000B 62		251	MOV T, A		119	
		252	*****		120	
		253	*****		121	
		254	*****		122	
		255	THE USER'S OWN TIMER INTERRUPT ROUTINE (IF IT EXISTS) COULD		123	
		256	BE PLACED AT THIS POINT		124	
		257	*****		125	
		258	*****		126	
		259	*****		127	
000C 1410		260	CALL REFRESH		128	
		261	*****		129	
		262	THE COMPLETE INTERRUPT ROUTINE SHOULD BE COPIED HERE		130	
		263	TO SAVE A FULL LEVEL OF SUBROUTINE NESTING.		131	
		264	IT WAS WRITTEN AS A SUBROUTINE HERE FOR THE SAKE OF CLARITY.		132	
		265	*****		133	
		266	*****		134	
		267	*****		135	
000E FA		268	TIRET TIMER INTERRUPT RETURN CODE- RESTORES ACC VALUE		136	
000F 93		269	TIRET MOV A, ASAVE		137	
		270	RETR		138	
		271	*****		139	
		272	#EJECT		140	
			*****		141	
			*****		142	
			*****		143	
			*****		144	
			*****		145	
			*****		146	
			*****		147	
			*****		148	
			*****		149	
			*****		150	
			*****		151	
			*****		152	
			*****		153	
			*****		154	
			*****		155	
			*****		156	
			*****		157	
			*****		158	
			*****		159	
			*****		160	
			*****		161	
			*****		162	
			*****		163	
			*****		164	
			*****		165	
			*****		166	
			*****		167	
			*****		168	
			*****		169	
			*****		170	
			*****		171	
			*****		172	
			*****		173	
			*****		174	
			*****		175	
			*****		176	
			*****		177	
			*****		178	
			*****		179	
			*****		180	
			*****		181	
			*****		182	
			*****		183	
			*****		184	
			*****		185	
			*****		186	
			*****		187	
			*****		188	
			*****		189	
			*****		190	
			*****		191	
			*****		192	
			*****		193	
			*****		194	
			*****		195	
			*****		196	
			*****		197	
			*****		198	
			*****		199	
			*****		200	
			*****		201	
			*****		202	
			*****		203	
			*****		204	
			*****		205	
			*****		206	
			*****		207	
			*****		208	
			*****		209	
			*****		210	
			*****		211	
			*****		212	
			*****		213	
			*****		214	
			*****		215	
			*****		216	
			*****		217	
			*****		218	
			*****		219	
			*****		220	
			*****		221	
			*****		222	
			*****		223	
			*****		224	
			*****		225	
			*****		226	
			*****		227	
			*****		228	
			*****		229	
			*****		230	
			*****		231	
			*****		232	

I815-II MCS-48/UPI-41 MACRO ASSEMBLER, V2.01

PAGE 5 815II22A.DOC 11-11-77

AP40: INTEL MCS-48 KEYBOARD/DISPLAY APPLICATION NOTE APPENDIX I

LOC	OBJ	SEQ	SOURCE STATEMENT
		273	*****
		274	REFRSH SUBROUTINE TO MULTIPLEX SEVEN-SEGMENT DISPLAYS
		275	EACH CALL CAUSES THE NEXT CHARACTER TO BE DISPLAYED.
		276	ACCORDING TO THE CONTENTS OF THE SEGMAP REGISTER ARRAY
		277	REFRSH SHOULD BE CALLED AT LEAST EVERY MSEC OR SO
		278	*****
		279	*****
0010	2300	280	REFRSH: MOV A,#BLANK XOR SEGPOLE
0012	39	281	OUTL PSGMNT,A ;WRITE BLANK PATTERN TO SEG DRIVERS
0013	2357	282	REFR1: MOV A,#CHRSTB ;LOOK UP DIGIT ENABLE PATTERN
0015	6F	283	ADD A,CURDIG ;ADD CURDIG DISPLACEMENT
0016	A3	284	MOV A,0A ;ENABLE ONE BIT OF ACCUMULATOR
0017	02	285	OUTL PDIGIT,A ;ENERGIZE CHARACTER
		286	;
		287	;
0018	2337	288	MOV A,#SEGMAP ;LOAD BASE OF REGISTER ARRAY
001A	6F	289	ADD A,CURDIG ;ADD CURDIG DISPLACEMENT
001B	A9	290	MOV PNTRL,A ;
001C	F1	291	MOV A,@PNTRL ;LOAD ACC W/ NEXT SEGMENT PATTERN
001D	39	292	OUTL PSGMNT,A ;ENABLE APPROPRIATE SEGMENTS
		293	;
		294	*****
		295	;
		296	THE NEXT CHARACTER IS NOW BEING DISPLAYED.
		297	THE KEYBOARD SCAN ROUTINE IS INTEGRATED INTO THE DISPLAY SCAN.
		298	WITH THE CURRENT ROW ENERGIZED, CHECK IF THERE ARE ANY INPUTS
		299	*****
001E	8821	300	SCAN: MOV PNTR0,#KEYLOC ;SET POINTER FOR SEVERAL KEYLOC REFERENCES
0020	0A	301	IN A,PINPUT ;LOAD ANY SWITCH CLOSURES
		302	;
		303	*****
		304	;; THIS BLOCK OF CODE IS NOT NEEDED BY THE KEYBOARD SCAN LOGIC. ;;
		305	;; HOWEVER, ITS INCLUSION WOULD SPEED THINGS UP A BIT BY ;;
		306	;; SKIPPING OVER ROWS IN WHICH NO KEYS ARE DOWN. ;;
		307	;; IT WAS OMITTED HERE TO CONSERVE ROM SPACE, BUT MIGHT BE ;;
		308	;; RESTORED IF VERY LARGE KEYBOARDS (ESPECIALLY THOSE WITH EIGHT ;;
		309	;; KEYS PER ROW) ARE TO BE USED WITH THIS ALGORITHM. ;;
		310	*****
		311	;; CPL A ;ANY CLOSURES DETECTED ARE NOW ONE BITS.;;
		312	;; ANL A,#INPMASK ;;
		313	;; JNZ SCAN1 ;--IF A KEY IN THE CURRENTLY ENABLED ROW IS DOWN ;;
		314	;; NO KEY IS NOW DOWN SO THE KEYLOC COUNT MAY BE UPDATED DIRECTLY ;;
		315	;; MOV A,@PNTR0 ;;
		316	;; ADD A,#NCOLS ;;
		317	;; MOV @PNTR0,A ;;
		318	;; JMP SCAN5 ;;
		319	*****
		320	;; IF THIS CODE IS USED, SUBSTITUTE THE 'JC SCAN5' FOUR LINES ;;
		321	;; HENCE WITH 'JNC SCAN5' TO ACCOMMODATE THE INVERTED POLARITY ;;
		322	*****
		323	\$EJECT

```

LOC  OBJ      SEQ      SOURCE STATEMENT
-----
324 ; *****
325 ; ROTATE BITS THROUGH THE CY WHILE INCREMENTING KEYLOC.
326 ; *****
327 ; *****
0021 BD04 328 SCAN1: MOV     R0CNT, #NCOLS ; SET UP FOR <NCOLS> LOOPS THROUGH 'NXTLOC'
0023 F7   329 NXTLOC: RLC     A ;
0024 AC   330     MOV     ROTPAT, A ; SAVE SHIFTED BIT PATTERN
0025 F63F 331     JC      SCANS5 ; ONE BIT IN CY INDICATES KEY NOT DOWN
332 ; *****
333 ; *****
334 ; *****
335 ; AT THIS POINT IT HAS JUST BEEN DETERMINED THAT THE VALUE
336 ; OF KEYLOC IS THE POSITION OF A KEY WHICH IS NOW DOWN
337 ; THE FOLLOWING CODE DEBOUNCES THE KEY, ETC.
338 ; IF MODIFICATIONS TO THE KEYBOARD LOGIC, I.E. THE INCLUSION
339 ; OF A SHIFT, CONTROL, OR MODE KEY IN THE KEY MATRIX ITSELF)
340 ; ARE DESIRED, THEY SHOULD BE MADE AT THIS POINT, BEFORE
341 ; THE DEBOUNCE LOGIC BEGINS. FOR EXAMPLE, AT THIS POINT
342 ; KEYLOC COULD BE COMPARED AGAINST THE POSITION OF THE MODE
343 ; KEY, AND IF THEY MATCH SET SOME FLAG BIT AND JUMP TO
344 ; LABEL 'SCANS' OR, BY COMPARING KEYLOC AGAINST THE LAST
345 ; KEY DEBOUNCED, IMMEDIATE TWO-KEY ROLLOVER COULD BE
346 ; IMPLEMENTED.
347 ; *****
348 ; *****
349 ; *****
0027 A5   350     CLR     F1 ; MARK THAT AT LEAST ONE KEY WAS DETECTED
0028 B5   351     CPL     F1 ; IN THE CURRENT SCAN
352 ; *****
353 ; *****
354 ; A KEYSTROKE WAS DETECTED FOR THE CURRENT COLUMN. ITS
355 ; POSITION IS IN REGISTER KEYLOC. SEE IF SAME KEY SENSED LAST CYCLE
356 ; *****
357 ; *****
0029 F0   358     MOV     A, @PNTR0 ; PNTR0 STILL HOLDS #KEYLOC
002A 2E   359     XCH     A, LASTKY
002B DE   360     XRL     A, LASTKY
002C B820 361     MOV     PNTR0, #NREPTS ; PREPARE TO CHECK AND/OR MODIFY REPEAT COUNT
002E C634 362     JZ      SCANS
363 ; *****
364 $EJECT

```

ISIS-II MCS-48/UPI-41 MACRO ASSEMBLER, V2.0

PAGE 10

AP40: INTEL MCS-48 KEYBOARD/DISPLAY APPLICATION NOTE APPENDIX

LOC	OBJ	SEQ	SOURCE STATEMENT	THEM/TAT2	30RU02	093	001
		365	*****				
		366	A DIFFERENT KEY WAS READ ON THIS CYCLE THAN ON THE PREVIOUS CYCLE.				
		367	SET NREPTS TO THE DEBOUNCE PARAMETER FOR A NEW COUNTDOWN				
		368	*****				
0030	B004	369					
0032	043F	370	MOV @PNTR0, #DEBNCE				
		371	JMP SCANS				
		372	;				
		373	*****				
		374	SAME KEY WAS DETECTED AS ON PREVIOUS CYCLE				
		375	LOOK AT NREPTS. IF ALREADY ZERO, DO NOTHING				
		376	ELSE DECREMENT NREPTS				
		377	IF THIS RESULTS IN ZERO, MOVE LASTKY INTO KDBUF				
		378	*****				
0034	F0	380	SCANS MOV A, @PNTR0				
0035	C63F	381	JZ SCANS ; IF ALREADY ZERO				
0037	07	382	DEC A ; INDICATE ONE MORE SUCCESSIVE KEY DETECTION				
0038	A0	383	MOV @PNTR0, A				
0039	963F	384	JNZ SCANS ; IF DECREMENT DOES NOT RESULT IN ZERO				
003B	FE	385	MOV A, LASTKY				
003C	B822	386	MOV PNTR0, #KDBUF				
003E	A0	387	MOV @PNTR0, A ; MARK NEW KEY CLOSURE				
		388	;				
003F	B821	389	SCANS MOV PNTR0, #KEYLOC				
0041	10	390	INC @PNTR0				
0042	FC	391	MOV A, ROTPAT				
0043	ED23	392	DJNZ ROTCNT, NXTLOC				
		393	;				
		394	;				
0045	EF57	395	SCANS DJNZ CURDIG, SCANS				
		396	;				
		397	\$EJECT				

ISTS-II MCS-48/UPI-41 MACRO ASSEMBLER V2.0

PAGE 11

AP40: INTEL MCS-48 KEYBOARD/DISPLAY APPLICATION NOTE APPENDIX

```

LOC OBJ    SEQ    SOURCE STATEMENT
398 ;
399 ;*****
400 ; THE FOLLOWING CODE SEGMENT IS USED BY THE KEYBOARD SCANNING ROUTINE
401 ; IT IS EXECUTED ONLY AFTER A REFRESH SEQUENCE OF ALL
402 ; THE CHARACTERS IN THE DISPLAY IS COMPLETED
403 ;*****
404 ;
0047 BF08    405     MOV     CURDIG,#CHARNO
0049 B000    406     MOV     @PNTR0,#0      ;PNTR0 STILL CONTAINS #KEYLOC
004B 764F    407     JF1      SCAN8      ;JUMP IF ANY KEYS WERE DETECTED
004D BEFF    408     MOV     LASTKY,#0FFH   ;CHANGE <LASTKY> WHEN NO KEYS ARE DOWN
004F A5      409     SCANS: CLR     F1
410 ;
411 ;*****
412 ; THE NEXT CODE SEGMENT IS THE INTERRUPT-DRIVEN PORTION OF THE 'DELAY'
413 ; UTILITY. IT DECREMENTS RAM LOCATION 'RDELAY' ONCE PER DISPLAY SCAN
414 ; IF 'RDELAY' IS NOT ALREADY ZERO
415 ;*****
0050 B923    417     MOV     PNTR1,#RDELAY
0052 F1      418     MOV     A,@PNTR1
0053 C657    419     JZ      SCAN9
0055 07      420     DEC     A
0056 A1      421     MOV     @PNTR1,A
422 ;
0057 83      423     SCAN9: RET
424 ;
425 ;*****
426 ;
427 ;CHRSTB IS THE BASE FOR THE PATTERNS TO ENABLE ONE-OF-CHARNO CHARACTERS.
0057      428     CHRSTB EQU     (1) AND 0FFH
0058 01      429     DB      (00000010 XOR CHRPOL)
0059 02      430     DB      (00000100 XOR CHRPOL)
005A 04      431     DB      (00001000 XOR CHRPOL)
005B 08      432     DB      (00010000 XOR CHRPOL)
005C 10      433     DB      (00100000 XOR CHRPOL)
005D 20      434     DB      (01000000 XOR CHRPOL)
005E 40      435     DB      (10000000 XOR CHRPOL)
005F 80      436     DB      (10000000 XOR CHRPOL)
437 ;
438 $EJECT

```

LOC	OBJ	SEQ	SOURCE STATEMENT	MACRO STATEMENT
		439	INIT	INITIALIZES PROCESSOR REGISTERS
0060	D5	440	INIT	SEL R01
0061	BF08	441	MOV	CURDIG, #CHARNO
0062	B322	442	MOV	PNTRO, #KBDUF
0063	B0FF	443	MOV	@PNTRO, #0FFH
0064	B821	444	MOV	PNTRO, #KEYLOC
0065	B000	445	MOV	@PNTRO, #0
0066	23F0	446	MOV	A, #INPMASK
006D	3A	447	OUTL	PINPUT, A ; SET BIDIRECTIONAL INPUT LINES
006E	C5	448	SEL	R00
006F	149E	449	CALL	CLEAR ; UTILITY FOR SETTING INITIAL DISPLAY REGISTERS.
0071	A5	450	CLR	F1
0072	23F0	451	MOV	A, #TICK ; LOAD INTERRUPT RATE VALUE
0074	62	452	MOV	T, A ; T, A TICK
0075	55	453	STRI	T, A ; T, A TICK
0076	25	454	EN	TONTI ; ENABLE TIMER INTERRUPTS
		455		
		456		
		457	*****	
		458		
		459	ECHO	CHECK FOR ANY NEW KEYSTROKES DETECTED
		460		TRANSLATE EACH KEYSTROKE INTO A SEGMENT PATTERN
		461		AND WRITE IT INTO THE APPROPRIATE DISPLAY REGISTER
		462		
		463	*****	
		464		
0077	1483	465	ECHO	CALL KBDIN ; GET NEXT KEYSTROKE
0079	B281	466	JBS	FKEY ; JUMP IF KEY IN RIGHAND COLUMN
		467		SINCE THE ACC IS USED BY ENRACC AND RENTRY, ITS CONTENTS MUST
		468		BE PROCESSED OR SAVED BEFORE ENRACC IS CALLED
007B	148A	469	CALL	ENRACC ; FORM APPROPRIATE SEGMENT PATTERN
007D	14DB	470	CALL	RENTRY ; WRITE PATTERN INTO DISPLAY REGISTERS
007F	0477	471	JMP	ECHO ; LOOP INDEFINITELY
		472		
0081	2400	473	FKEY	JMP FUNCTN ; JUMP TO OFF-PAGE CODE TO CALL DEMO ROUTINE
		474		
		475	#EJECT	

IS15-11 MCS-48/UFI-41 MACRO ASSEMBLER: V2.0

PAGE 13

AP40 INTEL MCS-48 KEYBOARD/DISPLAY APPLICATION NOTE APPENDIX

```

LOC  OBJ      SEG      SOURCE STATEMENT      STATEMENT      EPC      LAB      OBJ
476 *****
477
478 THE FOLLOWING SUBROUTINES IMPLEMENT THE UTILITIES COMMONLY USED FOR
479 MOST KEYBOARD/DISPLAY APPLICATIONS.
480 THEY COULD BE USED EXACTLY AS SHOWN HERE OR ADAPTED FOR SPECIAL CASES.
481 *****
482 *****
483
484 KBDIN KEYBOARD INPUT SUBROUTINE.
485 COULD BE USED TO INTERFACE THE USER'S BACKGROUND PROGRAM WITH
486 THE INTERRUPT DRIVEN KEYBOARD SCANNER.
487 RETURNS ONLY AFTER A NEW KEYSTROKE HAS BEEN DETECTED AND DEBOUNCED.
488 ENCODED VALUE OF KEY (RATHER THAN ITS POSITION IN SWITCH MATRIX) IS
489 RETURNED IN THE ACCUMULATOR
0083 B922 490 KBDIN MOV     PNTR1, #KBDBUF
0085 2380 491 MOV     A, #00H ; KBDUF WILL BE MARKED AS CLEAR
0087 21 492 XCH     A, #PNTR1 ; LOAD BUFFER VALUE
0088 F283 493 JG7     KBDIN *****
008A 038E 494 ADD     A, #LEGND5 ; ADD BASE OF KEY ENCODING TABLE
008C A3 495 MOVB    A, #A ; OBTAIN BYTE REPRESENTING KEY SIGNIFICANCE
008D 83 496 RET
497
498
499 LEGND5 IS THE BASE FOR TABLE SHOWING KEY MATRIX SIGNIFICANCE
500 FOR THE KEYBOARD USED IN THE PROTOTYPE.
501 KEY LAYOUT IS AS SHOWN TO THE RIGHT
502
503 NOTE THAT BIT6-BIT4 MAY BE USED TO ENCODE KEY TYPE. IN THIS CASE.
504 BIT4 INDICATES REGULAR DECIMAL DIGITS.
505 BIT5 INDICATES RIGHT-COLUMN FUNCTION KEYS.
506 BIT6 INDICATES PUNCTUATION MARKS ( * AND # ).
507
008E 508 LEGND5 EQU    ($ AND 0FFH) ; USE LOW ORDER BITS AS TABLE INDEX.
008E 4F 509 DB      4FH
008F 10 510 DB      10H
0090 4E 511 DB      4EH
0091 28 512 DB      28H ; PDIGIT4== 1 2 3 <1>
0092 17 513 DB      17H
0093 18 514 DB      18H ; PDIGIT5== 4 5 6 <2>
0094 19 515 DB      19H
0095 24 516 DB      24H ; PDIGIT6== 7 8 9 <3>
0096 14 517 DB      14H
0097 15 518 DB      15H ; PDIGIT7== * 0 # <4>
0098 16 519 DB      16H
0099 22 520 DB      22H ; ! ! ! !
009A 11 521 DB      11H ; ! ! ! !
009B 12 522 DB      12H ; V V V V
009C 13 523 DB      13H ; PINUT7 PINUT6 PINUT5 PINUT4
009D 21 524 DB      21H
525 $EJECT

```

ISIS-11 MCS-48/UPI-41 MICRO ASSEMBLER, V2.0 PAGE 14 OF 14
 AP40 INTEL MCS-48 KEYBOARD/DISPLAY APPLICATION NOTE APPENDIX

LOC	OBJ	SEQ	SOURCE STATEMENT	INSTR	ADDR	DISP	LEN	VAL
		526	*****					
		527						
		528	CLEAR WRITES 'BLANK' CHARACTERS INTO ALL DISPLAY REGISTERS.					
		529	RETURNS WITH NEXTPL SET TO LEFTMOST CHARACTER POSITION					
		530	FILL WRITES SEGMENT PATTERN NOW IN ACC INTO ALL DISPLAY REGISTERS					
009E	2000	531	CLEAR MOV A, #BLANK XOR SEGPOL					
00A0	B938	532	FILL MOV PNTR1, #SEGMAP+1					
00A2	BF08	533	MOV NEXTPL, #CHARNO					
00A4	A1	534	CLR1 MOV @PNTR1, A STORE THE BLANK CODE					
00A5	19	535	INC PNTR1 POINT TO NEXT CHARACTER TO THE LEFT					
00A6	EFA4	536	DJNZ NEXTPL, CLR1					
00A8	BF08	537	MOV NEXTPL, #CHARNO					
00AA	83	538	RET					
		539						
		540	*****					
		541						
		542	PRINT SUBROUTINE TO COPY A STRING OF BIT PATTERNS FROM ROM TO THE					
		543	DISPLAY REGISTERS. STRING STARTS AT LOCATION POINTED TO BY PNTR0.					
		544	CONTINUES UNTIL AN ESCAPE CODE (OFFH) IS REACHED.					
		545	NOTE THAT THE CHARACTER STRING PUT OUT MUST BE LOCATED ON THE SAME					
		546	PAGE AS THIS SUBROUTINE, SINCE SAME-PAGE MOVES ARE USED.					
		547	PRINT IN TURN CALLS EITHER SUBROUTINE 'WDISP' OR 'RENTY'					
		548	TO ACTUALLY EFFECT WRITING INTO THE DISPLAY REGISTERS.					
00AB	F8	549	PRINT: MOV A, PNTR0 LOAD NEXT CHARACTER LOCATION					
00AC	A3	550	MOVP A, @A LOAD BIT PATTERN INDIRECT					
00AD	C6B4	551	JZ PRNT1 ESCAPE PATTERN					
00AF	14D0	552	CALL WDISP OUTPUT TO NEXT CHARACTER POSITION					
		553	## CALL RENTRY INSTEAD IF MESSAGE IS TO BE RIGHT JUSTIFIED					
00B1	18	554	INC PNTR0 INDEX POINTER					
00B2	04AB	555	JMP PRINT					
00B4	83	556	PRNT1 PET DONE					
		557						
		558	*****					
		559						
		560	JOHN ARRAY HOLDS THE BIT PATTERNS FOR THE LETTERS 'JOHN' (SEE 'TEST2')					
		561	(NOTE THAT 'JOHN' IS WRITTEN IN LOWER CASE LETTERS)					
00B5		562	JOHN EQU \$ AND OFFH					
00B5	1E	563	DB 00011110B XOR SEGPOL					
00B6	5C	564	DB 01011100B XOR SEGPOL					
00B7	74	565	DB 01101000B XOR SEGPOL					
00B8	54	566	DB 01010100B XOR SEGPOL					
00B9	00	567	DB 00					
		568						
		569	EJECT					

ISIS-II MCS-48/UPI-41 MACRO ASSEMBLER, V2.0

PAGE 15

AP40: INTEL MCS-48 KEYBOARD/DISPLAY APPLICATION NOTE APPENDIX

LOC	OBJ	SEQ	SOURCE STATEMENT
		570	*****
		571	:
		572	ENCACC ENCODES LSNIBBLE OF ACC INTO HEX BIT PATTERN INTO ACC
00BA	530F	573	ENCACC ANL A, #ENCMASK
00BC	03C0	574	ADD A, #DGPATS
00BE	A3	575	MOVP A, @A
00BF	83	576	RET
		577	DGPATS IS THE BASE FOR THE TABLE OF SEGMENT PATTERNS FOR THE BASIC
		578	DIGITS HERE THE FULL HEX SET (0-F) IS INCLUDED.
		579	FOR MANY USER APPLICATIONS, THE CHARACTER SET MAY BE AMENDED OR AUGMENTED
		580	TO INCLUDE ADDITIONAL SPECIAL PURPOSE PATTERNS.
		581	FORMAT IS PGFEDCBA IN STANDARD SEVEN-SEGMENT ENCODING CONVENTION
		582	WHERE P REPRESENTS THE DECIMAL POINT
00C0		583	DGPATS EQU \$ AND 0FFH
00C0	3F	584	DB 00111111B XOR SEGPOL
00C1	06	585	DB 00000110B XOR SEGPOL
00C2	58	586	DB 01010111B XOR SEGPOL
00C3	4F	587	DB 01001111B XOR SEGPOL
00C4	66	588	DB 01100110B XOR SEGPOL
00C5	6D	589	DB 01101101B XOR SEGPOL
00C6	7D	590	DB 01111101B XOR SEGPOL
00C7	07	591	DB 00000111B XOR SEGPOL
00C8	7F	592	DB 01111111B XOR SEGPOL
00C9	67	593	DB 01100111B XOR SEGPOL
00CA	77	594	DB 01110111B XOR SEGPOL
00CB	7C	595	DB 01111100B XOR SEGPOL
00CC	39	596	DB 00111001B XOR SEGPOL
00CD	5E	597	DB 01011110B XOR SEGPOL
00CE	79	598	DB 01111001B XOR SEGPOL
00CF	71	599	DB 01110001B XOR SEGPOL
		600	INDEX POINTER
		601	*****
		602	:
		603	WDISP WRITES BIT PATTERN NOW IN ACC INTO NEXT CHARACTER POSITION
		604	OF THE DISPLAY (NEXTPL). ADJUSTS NEXTPL POINTER VALUE.
		605	RESULTS IN DISPLAY BEING FILLED LEFT TO RIGHT, THEN RESTARTING
00D0	A9	606	WDISP MOV PNTRL, A
00D1	FF	607	MOV A, NEXTPL
00D2	0337	608	ADD A, #SEGMAP
00D4	29	609	XCH A, PNTRL
00D5	A1	610	MOV @PNTRL, A
00D6	EFDA	611	DJNZ NEXTPL, WDISP1
00D8	BF08	612	MOV NEXTPL, #CHARNO
00DA	83	613	WDISP1 RET
		614	:
		615	*EJECT

1515-II MCS-48/UF1-41 MACRO ASSEMBLER V2.0

PAGE 1316

AP40 INTEL MCS-48 KEYBOARD/DISPLAY APPLICATION NOTE APPENDIX

LOC	OBJ	SEQ	SOURCE STATEMENT
		616	*****
		617	
		618	RENTY SUBROUTINE TO ENTER ACC CONTENTS INTO THE RIGHTMOST DIGIT
		619	AND SHIFT EVERYTHING ELSE ONE PLACE TO THE LEFT
000B	B938	620	RENTY: MOV PNTR1, #SEGMAP+1
000D	BF08	621	MOV NEXTPL, #CHARNO
000F	21	622	RENT1: XCH A, @PNTR1
00E0	19	623	INC PNTR1
00E1	EFDF	624	DJNZ NEXTPL, RENT1
00E2	BF08	625	MOV NEXTPL, #CHARNO ; POINT TO LEFTMOST CHARACTER
00E5	83	626	RET
		627	
		628	*****
		629	
		630	RDPOD TOGGLE DECIMAL POINT IN LAST CHARACTER/ DISPLAY CHARACTER
		631	DPADD TOGGLES DECIMAL POINT IN THE CHARACTER POINTED TO BY THE ACC
		632	
00E6	2301	633	RDPOD: MOV A, #01H ; SET INDEX TO RIGHTMOST POSITION
00E8	0337	634	DPADD: ADD A, #SEGMAP ; ACCESS DISPLAY REGISTER FOR DESIRED PLACE
00EA	A9	635	MOV PNTR1, A
00EB	F1	636	MOV A, @PNTR1
00EC	D390	637	XRL A, #00H
00EE	A1	638	MOV @PNTR1, A
00EF	83	639	RET
		640	
		641	*****
		642	
		643	HOLD SUBROUTINE CALLED WHEN KEY IS KNOWN TO BE DOWN.
		644	WILL NOT RETURN UNTIL KEY IS RELEASED.
00F0	05	645	HOLD: SEL RB1
00F1	FE	646	MOV A, LASTKY ; (LASTKY)=0FFH IFF NO KEYS DOWN
00F2	05	647	SEL RB0
00F3	37	648	CPL A
00F4	96F0	649	JNZ HOLD
00F6	83	650	RET
		651	
		652	*****
		653	
		654	DELAY SUBROUTINE HANGS UP FOR THE NUMBER OF COMPLETE DISPLAY SCANS EQUAL
		655	TO THE CONTENTS OF THE ACCUMULATOR WHEN CALLED.
00F7	B923	656	DELAY: MOV PNTR1, #DELAY
00F9	A1	657	MOV @PNTR1, A
00FA	F1	658	DELAY1: MOV A, @PNTR1
00FB	96FA	659	JNZ DELAY1
00FD	83	660	RET
		661	#EJECT

ISIS-II MCS-48/UP1-41 MACRO ASSEMBLER V2.0

PAGE 17

AP40: INTEL MCS-48 KEYBOARD/DISPLAY APPLICATION NOTE APPENDIX: VAL210

```

LOC  OBJ      SEQ      SOURCE STATEMENT      THIRDTATE 30002      000      100  001
0100      662 ORG      100H
0101      663
0102      664 *****
0103      665
0104      666 THE CODE ON THIS PAGE IS FOR DEMONSTRATION PURPOSES ONLY-
0105      667 I TRULY DOUBT WHETHER ANY END USERS WOULD LIKE TO SEE A NAME
0106      668 POPPING UP ON THEIR CALCULATOR SCREENS.
0107      669 HOWEVER, THE CODE SHOWN HERE DOES INDICATE HOW THE UTILITY SUBROUTINES
0108      670 INCLUDED HERE COULD BE ACCESSED.
0109      671 THE ROUTINES THEMSELVES ARE CALLED WHEN ONE OF THE FOUR BUTTONS
0110      672 ON THE RIGHT-HAND SIDE OF THE PROTOTYPE KEYBOARD IS PRESSED.
0111      673
0112      674 *****
0113      675
0114      676 FUNCTN ROUTINE TO IMPLEMENT ONE OF FOUR DEMO UTILITIES, ACCORDING
0115      677 TO WHICH OF THE FOUR FUNCTION KEYS WAS PRESSED
0116      678 FUNCTN: JB0      FUNCT1
0117      679 JB1      FUNCT2
0118      680 JB2      FUNCT3
0119      681
0120      682 FUNCT4: CALL    RDPADD
0121      683 JMP      ECHO
0122      684
0123      685 FUNCT3: CALL    TEST3
0124      686 JMP      ECHO
0125      687 *****
0126      688 FUNCT2: CALL    TEST2
0127      689 JMP      ECHO
0128      690
0129      691 FUNCT1: CALL    TEST1
0130      692 JMP      ECHO
0131      693
0132      694 *****
0133      695
0134      696 TEST1 CODE SEGMENT TO FILL DISPLAY REGISTERS WITH DIGITS DOWN TO '1'
0135      697 TEST1: MOV      NEXTPL, #CHARNO
0136      698 MOV      PNTR0, #CHARNO ; SET FOR EIGHT LOOP REPETITIONS
0137      699 TST11: MOV      A, NEXTPL
0138      700 CALL    ENCRCC
0139      701 CALL    NDISP
0140      702 D.JNZ    PNTR0, TST11 ; COPY NEXT DIGIT INTO DISPLAY REGISTERS
0141      703 MOV      NEXTPL, #CHARNO
0142      704 RET
0143      705
0144      706 $EJECT

```


1-48

ROTCNT	204#	328	392															
ROTPAT	203#	330	391															
SCAN	300#																	
SCAN1	328#																	
SCAN3	362	380#																
SCAN5	331	371	381	384	389#													
SCAN6	395#																	
SCAN8	407	409#																
SCAN9	395	419	423#															
SEGMAP	220#	288	532	608	620	634												
SEGPOL	170#	280	531	563	564	565	566	584	585	566	587	588	589	590	591	592		
	593	594	595	596	597	598	599	722										
TEST1	691	697#																
TEST2	688	711#																
TEST3	685	722#																
TICK	177#	250	451															
TIINT	248#																	
TIRET	269#																	
TST11	699#	702																
WDISP	552	606#	701															
WDISP1	611	613#																

CROSS REFERENCE COMPLETE



APPLICATION NOTE

AP-49

January 1979

Serial I/O and Math Utilities for the 8049 Microcomputer

Lionel Smith and Cecil Moore
Microcomputer Applications

INTRODUCTION

The Intel® MCS-48 family of microcomputers marked the first time an eight bit computer with program storage, data storage, and I/O facilities was available on a single LSI chip. The performance of the initial processors in the family (the 8748 and the 8048) has been shown to meet or exceed the requirements of most current applications of microcomputers. A new member of the family, however, has been recently introduced which promises to allow the use of the single chip microcomputer in many application areas which have previously required a multichip solution. The Intel® 8049 virtually doubles processing power available to the systems designer. Program storage has been increased from 1K bytes to 2K bytes, data storage has been increased from 64 bytes to 128 bytes, and processing speed has been increased by over 80%. (The 2.5 microsecond instruction cycle of the first members of the family has been reduced to 1.36 microseconds.)

It is obvious that this increase in performance is going to result in far more ambitious programs being written for execution in a single chip microcomputer. This article will show how several program modules can be designed using the 8049. These modules were chosen to illustrate the capability of the 8049 in frequently encountered design situations. The modules included are full duplex serial I/O, binary multiply and divide routines, binary to BCD conversions, and BCD to binary conversion. It should be noted that since the 8049 is totally software compatible with the 8748 and 8048 these routines will also be useful directly on these processors. In addition the algorithms for these programs are expressed in a program design language format which should allow them to be easily understood and extended to suit individual applications with minimal problems.

FULL DUPLEX SERIAL COMMUNICATIONS

Serial communications have always been an important facet in the application of microprocessors. Although this has been partially due to the necessity of connecting a terminal to the microprocessor based system for program generation and debug, the main impetus has been the simple fact that a large share of microprocessors find their way into end products (such as intelligent terminals) which themselves depend on serial communication. When it is necessary to add a serial link to a microprocessor such as the Intel® MCS-85 or 86 the solution is easy; the Intel® 8251A USART or 8273 SDLC chip can easily be added to provide the necessary protocol. When it is necessary to do the same thing to a single chip microcomputer, however, the situation becomes more difficult.

Some microcomputers, such as the Intel 8048 and 8049 have a complete bus interface built into them which allows the simple connection of a USART to the processor chip. Most other single chip microcomputers, although lacking such a bus, can be connected to a USART with various artificial hardware and software constructs. The difficulty with using these chips,

however, is more economic than technical; these same peripheral chips which are such a bargain when coupled to a microprocessor such as the MCS-85 or 86, have a significant cost impact on a single chip microcomputer based system. The high speed of the 8049, however, makes it feasible to implement a serial link under software control with no hardware requirements beyond two of the I/O pins already resident on the microcomputer.

There are many techniques for implementing serial I/O under software control. The application note "Application Techniques for the MCS-48 Family" describes several alternatives suitable for half duplex operation. Full duplex operation is more difficult, however, since it requires the receive and transmit processes to operate concurrently. This difficulty is made more severe if it is necessary for some other process to also operate while serial communication is occurring. Scanning a keyboard and display, for example, is a common operation of single chip microcomputer based system which might have to occur concurrently with the serial receive/transmit process. The next section will describe an algorithm which implements full duplex serial communication to occur concurrently with other tasks. The design goal was to allow 2400 baud, full duplex, serial communication while utilizing no more than 50% of the available processing power of the high speed 8049 microcomputer.

The format used for most asynchronous communication is shown in Figure 1. It consists of eight data bits with a leading 'START' bit and one or more trailing 'STOP' bits. The START bit is used to establish synchronization between the receiver and transmitter. The STOP bits ensure that the receiver will be ready to synchronize itself when the next start bit occurs. Two stop bits are normally used for 110 baud communication and one stop bit for higher rates.

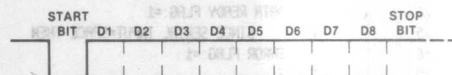


Figure 1.

The algorithm used for reception of the serial data is shown in Figure 2. It uses the on board timer of the 8049 to establish a sampling period of four times the desired baud rates. For 2400 baud operation a crystal frequency of 9.216 MHz was chosen after the following calculation:

$$f = 480N(2400/4)$$

where 480 is the factor by which the crystal frequency is divided within the processor to get the basic interrupt rate
2400 is the desired baud rate
4 is the required number of samples per bit time
N is the value loaded into the MCS-48 timer when it overflows

The value N was chosen to be two (resulting in $f = 9.216$ MHz) so that the operating frequency of the 8049 could be as high as possible without exceeding the maximum frequency specification of the 8049 (11 MHz).

```

START OF RECEIVE ROUTINE
=====
1 IF RECEIVE FLAG=0 THEN
2   IF SERIAL INPUT=SPACE THEN
3     RECEIVE FLAG:=1
4     BYTE FINISHED FLAG:=0
5   ENDIF
6 ELSE SINCE RECEIVE FLAG=1 THEN
7   IF SYNC FLAG=0 THEN
8     IF SERIAL INPUT=SPACE THEN
9       SYNC FLAG:=1
10      DATA:=00H
11      SAMPLE CNTR:=4
12    ELSE SINCE SERIAL INPUT=MARK THEN
13      RECEIVE FLAG:=0
14    ENDIF
15  ELSE SINCE SYNC FLAG=1 THEN
16    SAMPLE COUNTER:=SAMPLE COUNTER-1
17    IF SAMPLE COUNTER=0 THEN
18      SAMPLE COUNTER:=4
19    ENDIF
20  ENDIF
21  IF BYTE FINISHED FLAG=0 THEN
22    CARRY:=SERIAL INPUT
23    SHIFT DATA RIGHT WITH CARRY
24    IF CARRY=1 THEN
25      OKDATA:=DATA
26      IF DATA READY FLAG=0 THEN
27        BYTE FINISHED FLAG:=1
28      ELSE
29        BYTE FINISHED FLAG:=1
30        OVERRUN FLAG:=1
31      ENDIF
32    ENDIF
33  ELSE SINCE BYTE FINISHED FLAG=1 THEN
34    IF SERIAL INPUT=MARK THEN
35      DATA READY FLAG:=1
36    ELSE SINCE SERIAL INPUT=SPACE THEN
37      ERROR FLAG:=1
38    ENDIF
39    RECEIVE FLAG:=0
40    SYNC FLAG:=0
41  ENDIF
42 ENDIF
43 ENDIF
44 ENDIF

```

Figure 2

The timer interrupt service routine always loads the timer with a constant value. In effect the timer is used to generate an independent time base of four times the required baud rate. This time base is free running and is never modified by either the receive or transmit programs, thus allowing both of them to use the same timer. Routines which do other time dependent tasks (such as scanning keyboards) can also be called periodically at some fixed multiple of this basic time unit.

The algorithm shown in Figure 2 uses this basic clock plus a handful of flags to process the serial input data.

Once the meaning of these flags are understood the operation of the algorithm should be clear. The **Receive Flag** is set whenever the program is in the process of receiving a character. The **Synch Flag** is set when the center of the start bit has been checked and found to be a SPACE (if a MARK is detected at this point the receiver process has been triggered by a noise pulse so the program clears the **Receive Flag** and returns to the idle state). When the program detects synchronization it loads the variable **DATA** with 80H and starts sampling the serial line every four counts. As the data is received it is right shifted into variable **DATA**; after eight bits have been received the initial one set into **DATA** will result in a carry out and the program knows that it has received all eight bits. At this point it will transfer all eight bits to the variable **OKDATA** and set the **Byte Finished Flag** so that on the next sample it will test for a valid stop bit instead of shifting in data. If this test is successful the **Data Ready Flag** will be set to indicate that the data is available to the main process. If the test is unsuccessful the **Error Flag** will be set.

The transmit algorithm is shown in Figure 3. It is executed immediately following the receive process. It is a simple program which divides the free running clock down and transmits a bit every fourth clock. The variable **TICK COUNTER** is used to do the division. The **Transmitting Flag** indicates when a character transmission is in progress and is also used to determine when the **START** bit should be sent. The **TICK COUNTER** is used to determine when to send the next bit ($\text{TICK COUNTER MODULO } 4 = 0$) and also when the **STOP** bits should be sent ($\text{TICK COUNTER} = 9$). After the transmit routine completes any other timer based routines, such as a keyboard/display scanner or a real time clock, can be executed.

```

START OF TRANSMIT ROUTINE
=====
1 TICK COUNTER:=TICK COUNTER+1
2 IF TICK COUNTER MOD 4=0 THEN
3   IF TRANSMITTING FLAG=1 THEN
4     IF TICK COUNTER=00 1010 00 BINARY THEN
5       TRANSMITTING FLAG:=0
6     ELSE IF TICK COUNTER=00 1001 00 BINARY THEN
7       SEND END MARK
8       TRANSMITTING FLAG:=0
9     ELSE SINCE TICK COUNTER>THE ABOVE COUNT THEN
10      SEND NEXT BIT
11    ENDIF
12 ELSE SINCE TRANSMITTING FLAG=0 THEN
13   IF TRANSMIT REQUEST FLAG=1 THEN
14     TXBYT:=TXBYT
15     TRANSMIT REQUEST FLAG:=0
16     TRANSMITTING FLAG:=1
17     TICK COUNTER:=0
18   ENDIF
19   SEND SYNC BIT (SPACE)
20 ENDIF
21 ENDIF

```

Figure 3

Figure 4 shows the complete receive and transmit programs as they are implemented in the instruction set of

the 8049. Also included in Fig. 4 is a short routine which was used to test the algorithm.

ISIS-II MCS-48/UPI-41 MACRO ASSEMBLER, V2.0

```

LOC  OBJ      SEQ      SOURCE STATEMENT
1 ;*****
2 ;*
3 ;*      THIS PROGRAM TESTS THE FULL DUPLEX COMMUNICATION SOFTWARE
4 ;*
5 ;*****
6 ;
7 $INCLUDE('F1.URTEST.PDL')
8 ;
9      START OF TEST ROUTINE
10 ;
11 ;
12 ;
13 ;
14 ;
15 ;
16 ;1 ERROR COUNT =0
17 ;1 REPEAT
18 ;2   PATTERN =0
19 ;2   INITIALIZE TIMER
20 ;2   CLEAR FLAGBYTE
21 ;2   FLAG1=MARK
22 ;2   REPEAT
23 ;3   IF TRANSMIT REQUEST FLAG=0 THEN
24 ;4       N:TBYTE =PATTERN
25 ;4       TRANSMIT REQUEST FLAG=1
26 ;3   ENDIF
27 ;3   IF DATA READY FLAG=1 THEN
28 ;4       PATTERN =OKDATA
29 ;4       DATA READY FLAG =0
30 ;3   ENDIF
31 ;2   UNTIL ERROR FLAG OR OVERRUN FLAG
32 ;2   INCREMENT ERROR COUNT
33 ;1 UNTIL FOREVER
34 ; EOF
35 $EJECT
0000 36      ORG      0
37 ;1 SELECT REGISTER BANK 0
0000 C5 38      SEL     RB0
39 ;1 GOTO TEST
0001 2400 40      JMP     TEST
41 $      INCLUDE('F1.UART')
42 ;
43 ;
44 ; ASYNCHRONOUS RECEIVE/TRANSMIT ROUTINE
45 ;
46 ; THIS ROUTINE RECEIVES SERIAL CODE USING PIN 10 AS RXD
47 ; AND CONCURRENTLY TRANSMITS USING PIN P27
48 ; NOTE
49 ; THIS ROUTINE USES FLAG 1 TO BUFFER THE TRANSMITTED

```

Figure 4

```

LOC: OBJ: SRC: SOURCE STATEMENT:
= 50 ; 1 DATA LINE. THIS ELIMINATES THE JITTER THAT
= 51 ; 1 WOULD BE CAUSED BY VARIATIONS IN THE RECEIVE
= 52 ; 1 TIMING. NO OTHER PROGRAM MAY USE FLAG 1 WHILE
= 53 ; 1 THE TIMER INTERRUPT IS ENABLED
= 54 ;
= 55 ;
= 56 ;
= 57 ;
= 58 ;
= 59 ; REGISTER ASSIGNMENTS-BANK1
= 60 ; =====
= 61 ;
= 62 ;
0007 = 63 ATEMP EQU R7 ; USED TO SAVE ACCUMULATOR CONTENTS DURING INTERRUPT
0006 = 64 FLGBYT EQU R6 ; CONTAINS VARIOUS FLAGS USED TO CONTROL THE RECEIVE
= 65 ; AND TRANSMIT PROCESS. SEE CONSTANT DEFINITIONS FOR
= 66 ; THE MEANING OF EACH BIT
0005 = 67 SAMCTR EQU R5 ; SAMPLE COUNTER FOR THE RECIEVE PROCESS
0004 = 68 TCKCTR EQU R4 ; SAMPLE COUNTER FOR THE TRANSMIT PROCESS
0000 = 69 REG0 EQU R0 ; USED AS POINTER REGISTER
= 70 ;
= 71 ; RAM ASSIGNMENTS
= 72 ; =====
= 73 ;
0020 = 74 MOKDAT EQU 20H ; RECEIVE RETURNS VALID DATA IN THIS BYTE
0021 = 75 MDATA EQU 21H ; RECEIVE ACCUMULATES DATA IN THIS BYTE
0022 = 76 MXMTBY EQU 22H ; CONTAINS BYTE BEING TRANSMITTED
0023 = 77 MNXTBY EQU 23H ; CONTAINS THE NEXT BYTE TO BE TRANSMITTED
= 78 #REJECT
= 79 ;
= 80 ;
= 81 ; CONSTANTS
= 82 ; =====
= 83 ;
= 84 ; THE FOLLOWING CONSTANTS ARE USED TO ACCESS THE FLAG BITS CONTAINED
= 85 ; IN REGISTER FLGBYT
= 86 ;
0001 = 87 RCVFLG EQU 01H ; SET WHEN START BIT IS FIRST DETECTED
= 88 ; RESET WHEN RECEIVE PROCESS IS COMPLETE
0002 = 89 SYNFLG EQU 02H ; SET WHEN START BIT IS VERIFIED
= 90 ; RESET WHEN RECEIVE PROCESS IS COMPLETE
0004 = 91 BYFNFL EQU 04H ; RESET WHEN START BIT IS FIRST DETECTED
= 92 ; SET WHEN THE EIGHT DATA BITS HAVE ALL BEEN RECEIVED
0008 = 93 DRDYFL EQU 08H ; SHOULD BE RESET BY MAIN PROGRAM WHEN DATA IS ACCEPTED
= 94 ; SET BY RECEIVE PROCESS WHEN STOP BIT(S) ARE VERIFIED
0010 = 95 ERRFLG EQU 10H ; SHOULD BE RESET BY MAIN PROGRAM WHEN SAMPLED
= 96 ; SET BY RECEIVE PROCESS IF A FRAMING ERROR IS DETECTED
0020 = 97 TRROFL EQU 20H ; TESTED BY MAIN PROGRAM TO DETERMINE IF READY TO
= 98 ; TRANSMIT A NEW BYTE-SET TO INDICATE THAT NXTBYT
= 99 ; HAS BEEN LOADED
= 100 ; RESET BY TRANSMIT PROCESS WHEN BYTE IS ACCEPTED
0040 = 101 TRNGFL EQU 40H ; SET WHEN TRANSMISSION OF A BYTE STARTS
= 102 ; RESET WHEN STOP BIT IS TRANSMITTED
0080 = 103 OVRUN EQU 80H ; SET BY RECEIVE PROCESS WHEN OVERUN OCCURS
= 104 ; SHOULD BE RESET BY MAIN PROGRAM WHEN SAMPLED

```

Figure 4 (continued)

LOC	OBJ	SEQ	SOURCE STATEMENT
		= 105 ;	
		= 106 ;	GENERAL CONSTANTS
		= 107 ;	=====
		= 108 ;	
0000		= 109 MARK	EQU 80H ; USED TO GENERATE A MARK
FF7F		= 110 SPACE	EQU NOT 80H ; USED TO GENERATE A SPACE
0000		= 111 STPBTS	EQU 0 ; CONTROLS THE NUMBER OF STOP BITS
		= 112	0 GENERATES ONE STOP BIT
		= 113	1 GENERATES TWO STOP BITS
		= 114 ;	
		= 115 \$EJECT	
		= 116 ;	
		= 117 ;	START OF RECEIVE/TRANSMIT INTERRUPT SERVICE ROUTINE
		= 118 ;	=====
		= 119 ;	
0007		= 120	ORG 0007H
		= 121	
0007 160A		= 122	1 ENTER INTERRUPT MODE
0009 93		= 123 TISR:	JTF UART
000A D5		= 124	RETR
		= 125 UART:	SEL RB1
		= 126	1 SAVE ACCUMULATOR CONTENTS
000B AF		= 127	MOV ATEMP, A
		= 128	1 RELOAD TIMER
000C 23FE		= 129	MOV A, #TIMCNT
000E 62		= 130	MOV T, A
		= 131 ;	
		= 132 ;	OUTPUT TXD BUFFER (F1) TO TXD I/O LINE (P27)
		= 133 ;	=====
		= 134 ;	
000F 7615		= 135	JF1 OMARK
0011 9A7F		= 136 OSPACE:	ANL P2, #SPACE
0013 0417		= 137	JMP RCV000
0015 8A80		= 138 OMARK	ORL P2, #MARK
		= 139 ;	
		= 140 ;	START OF RECEIVE ROUTINE
		= 141 ;	=====
		= 142 ;	
		= 143	1 IF RECEIVE FLAG=0 THEN
0017 FE		= 144 RCV000:	MOV A, FLGBYT
0018 1224		= 145	JB0 RCV010
		= 146	2 IF SERIAL INPUT=SPACE THEN
001A 3664		= 147	JT0 XMIT
		= 148	3 RECEIVE FLAG:=1
001C FE		= 149	MOV A, FLGBYT
001D 4301		= 150	ORL A, #RCVFLG
		= 151	3 BYTE FINISHED FLAG:=0
001F 53FB		= 152	ANL A, #NOT BYFNFL
		= 153	2 ENDIF
0021 AE		= 154	MOV FLGBYT, A
0022 0464		= 155	JMP XMIT
		= 156	1 ELSE SINCE RECEIVE FLAG=1 THEN
		= 157	2 IF SYNC FLAG=0 THEN
0024 3238		= 158 RCV010:	JB1 RCV030
		= 159	3 IF SERIAL INPUT=SPACE THEN

Figure 4 (continued)

LOC	OBJ	SEQ	SOURCE STATEMENT
0026	3633	= 160	JT0 RCV020
		= 161 : 4	SYNC FLAG = 1
0028	4302	= 162	ORL A, #SYNFLAG
002A	AE	= 163	MOV FLGBYT, A
		= 164 : 4	DATA = 00H
002B	B821	= 165	MOV R0, #MDATA
002D	B080	= 166	MOV @R0, #00H
		= 167 : 4	SAMPLE CNTR = 4
002F	B004	= 168	MOV SAMCTR, #4
0031	0464	= 169	JMP XMIT
		= 170 : 3	ELSE SINCE SERIAL INPUT=MARK THEN
		= 171 : 4	RECEIVE FLAG = 0
0033	53FE	= 172 RCV020	ANL A, #NOT RCVFLG
		= 173 : 3	ENDIF
0035	AE	= 174	MOV FLGBYT, A
0036	0464	= 175	JMP XMIT
		= 176 : 2	ELSE SINCE SYNC FLAG=1 THEN
		= 177 : 3	SAMPLE COUNTER = SAMPLE COUNTER - 1
0038	ED64	= 178 RCV030	DJNZ SAMCTR, XMIT
		= 179 : 3	IF SAMPLE COUNTER=0 THEN
		= 180 : 4	SAMPLE COUNTER = 4
003A	B004	= 181	MOV SAMCTR, #4
		= 182 : 4	IF BYTE FINISHED FLAG=0 THEN
003C	5259	= 183	JB2 RCV050
003E	97	= 184	CLR C
		= 185 : 5	CARRY = SERIAL INPUT
003F	2642	= 186	JNT0 RCV040
0041	A7	= 187	CPL C
0042	B821	= 188 RCV040	MOV R0, #MDATA
0044	F0	= 189	MOV A, @R0
		= 190 : 5	SHIFT DATA RIGHT WITH CARRY
0045	67	= 191	RRC A
0046	A0	= 192	MOV @R0, A
		= 193 : 5	IF CARRY=1 THEN
0047	E664	= 194	JNC XMIT
		= 195 : 6	OKDATA = DATA
0049	B820	= 196	MOV R0, #OKDAT
004B	A0	= 197	MOV @R0, A
		= 198 : 6	IF DATA READY FLAG=0 THEN
004C	FE	= 199	MOV A, FLGBYT
004D	7254	= 200	JB3 RCV045
		= 201 : 7	BYTE FINISHED FLAG=1
004F	4304	= 202	ORL A, #BYFNFL
0051	AE	= 203	MOV FLGBYT, A
0052	0464	= 204	JMP XMIT
		= 205 : 6	ELSE
		= 206 : 7	BYTE FINISHED FLAG = 1
		= 207 : 7	OVERRUN FLAG = 1
		= 208 RCV045	
		= 209	MOV A, FLGBYT
0054	4304	= 210	ORL A, # (BYFNFL OR OVRUN)
0056	AE	= 211	MOV FLGBYT, A
		= 212 : 6	ENDIF
		= 213 : 5	ENDIF
0057	0464	= 214	JMP XMIT

Figure 4 (continued)

LOC	OBJ	SEQ	SOURCE STATEMENT	30002	032	00000
		= 215 ; 4	ELSE IF SINCE BYTE FINISHED FLAG=1 THEN			
		= 216 ; 5	JMPN TIF SERIAL INPUT=MARK THEN			
0059	265F	= 217 RCV050: JNT0	RCV060 VON	STS =		
		= 218 ; 6	WRITE DATA READY FLAG:=1	STS =		
005B	4308	= 219	ORL A, #RDYFLG	STS =		
005D	0461	= 220	JMP T3 RCV070	STS =		
		= 221 ; 5	ELSE SINCE SERIAL INPUT=SPACE THEN			
		= 222 ; 6	WRITE ERROR FLAG:=1	STS =		
005F	4310	= 223 RCV060: ORL	A, #ERRFLG	STS =		
		= 224 ; 5	ENDIF	STS =		
		= 225 ; 5	RECEIVE FLAG:=0	STS =		
		= 226 ; 5	SYNC FLAG:=0	STS =		
0061	53FC	= 227 RCV070: ANL	A, #NOT(SYNFLG OR RCVFLG)	STS =		
0063	AE	= 228	MOV FLGBYT, A	STS =		
		= 229 ; 4	ENDIF	STS =		
		= 230 ; 3	ENDIF	STS =		
		= 231 ; 2	ENDIF	STS =		
		= 232 ; 1	ENDIF	STS =		
		= 233	EJECT	STS =		
		= 234 ;		STS =		
		= 235 ;	START OF TRANSMIT ROUTINE	STS =		
		= 236 ;	=====	STS =		
		= 237 ;		STS =		
		= 238 ; 1		STS =		
		= 239	TRANSMITTER OUTPUT BIT IS P2-7	STS =		
		= 240 ; 1	TICK COUNTER:=TICK COUNTER+1	STS =		
0064	1C	= 241 XMIT: INC	TCKCTR	STS =		
		= 242 ; 1	IF TICK COUNTER MOD 4=0 THEN	STS =		
0065	2303	= 243	MOV A, #03H	STS =		
0067	5C	= 244	ANL A, TCKCTR	STS =		
0068	9697	= 245	JNZ RETURN	STS =		
		= 246 ; 2	IF TRANSMITTING FLAG=1 THEN	STS =		
006A	FE	= 247	MOV A, FLGBYT	STS =		
006B	37	= 248	CPL A	STS =		
006C	D286	= 249	JB6 XMT040	STS =		
		= 250	IF STPBTS EQ 1	STS =		
		= 251 ; 3	IF TICK COUNTER=00 1010 00 BINARY THEN	STS =		
		= 252	MOV A, #28H	STS =		
		= 253	XRL A, TCKCTR	STS =		
		= 254	JNZ XMT010	STS =		
		= 255 ; 4	TRANSMITTING FLAG:=0	STS =		
		= 256	MOV A, FLGBYT	STS =		
		= 257	ANL A, #NOT TRNGFL	STS =		
		= 258	MOV FLGBYT, A	STS =		
		= 259	RETURN	STS =		
		= 260	ENDIF	STS =		
		= 261 ; 3	ELSE IF TICK COUNTER=00 1001 00 BINARY THEN	STS =		
006E	2324	= 262 XMT010: MOV	A, #24H	STS =		
0070	DC	= 263	XRL A, TCKCTR	STS =		
0071	967B	= 264	JNZ XMT020	STS =		
		= 265 ; 4	SEND END MARK	STS =		
0073	AS	= 266	CLR F1	STS =		
0074	85	= 267	CPL F1	STS =		
		= 268	IF STPBTS EQ 0	STS =		
		= 269 ; 4	TRANSMITTING FLAG:=0	STS =		

Figure 4 (continued)

LOC	OBJ	SEQ	SOURCE STATEMENT	330002	032	100	00
0075 FE	ENT	= 270	MOV R0, A, FLGBYT				CONDITIONAL ASSEMBLY
0076 53BF		= 271	ANL A, A, #NOT TRNGFL				
0078 AE		= 272	MOV R0, FLGBYT, A				
0079 0497		= 273	JMP R0, RETURN				
		= 274	ENDIF				
		= 275 ; 3	ELSE SINCE TICK COUNTER > THE ABOVE COUNT THEN				
		= 276 ; 4	SEND NEXT BIT				
007B B822		= 277	XMT020: MOV R0, #XMTBY				
007D F0		= 278	MOV R0, R0				
007E 67		= 279	RRC A				
007F A0		= 280	MOV R0, A				
0080 A5		= 281	CLR F1 ; FLAG 1 WILL BE USED TO BUFFER TXD				
0081 E697		= 282	JNC RETURN ; GO TO RETURN POINT IF TXD=SPACE (0)				
0083 B5		= 283	CPL F1 ; ELSE COMPLEMENT FLAG 1 TO A MARK				
0084 0497		= 284	JMP RETURN				
		= 285 ; 3	ENDIF				
		= 286 ; 2	ELSE SINCE TRANSMITTING FLAG=0 THEN				
		= 287 ; 3	IF TRANSMIT REQUEST FLAG=1 THEN				
0086 B297		= 288	XMT040: JB5 RETURN ; FLAG BYTE THERE				
		= 289 ; 4	XMTBYT:=NXTBYT				
0088 B823		= 290	MOV R0, #NXTBYT				
008A F0		= 291	MOV R0, R0				
008B B822		= 292	MOV R0, #XMTBYT				
008D A0		= 293	MOV R0, A				
		= 294 ; 4	TRANSMIT REQUEST FLAG:=0				
008E FE		= 295	MOV R0, A, FLGBYT				
008F 53DF		= 296	ANL A, A, #NOT TRNGFL				
		= 297 ; 4	TRANSMITTING FLAG:=1				
0091 4340		= 298	ORL A, #TRNGFL				
0093 AE		= 299	MOV FLGBYT, A				
		= 300 ; 4	TICK COUNTER:=0				
0094 BC00		= 301	MOV TCKCTR, #0				
		= 302 ; 4	SEND SYNC BIT (SPACE)				
0096 A5		= 303	CLR F1 ; SET FLAG 1 TO CAUSE A SPACE				
		= 304 ; 3	ENDIF				
		= 305 ; 2	ENDIF				
		= 306 ; 1	ENDIF				
		= 307	RETURN				
		= 308 ; 1	RESTORE ACCUMULATOR				
0097 FF		= 309	MOV A, ATEMP				
0098 93		= 310	RETR				
		= 311	#EJECT				
		= 312 ;					
		= 313 ;	START OF TEST ROUTINE				
		= 314 ;	=====				
		= 315 ;					
0100		= 316	ORG 0100H				
FFFE		= 317	TIMCNT EQU -2				
001E		= 318	FLGBYT EQU 1EH				
001D		= 319	MSAMCT EQU 1DH				
001C		= 320	MTCKCT EQU 1CH				
		= 321 ;					
0007		= 322	ERRCNT EQU R7				
0006		= 323	PATT EQU R6				
		= 324 ;					

Figure 4 (continued)

LOC	OBJ	SEQ	SOURCE STATEMENT	OBJECT STATEMENT
		325 ;		
		326 ;		
0100	BF00	327 ;1	ERROR COUNT:=0	
		328	TEST: MOV ERRCNT, #0	
		329 ;1	REPEAT	
		330	TLOP:	
0102	BE00	331 ;2	PATTERN:=0	
		332	MOV PATT, #00	
0104	23FE	333 ;2	INITIALIZE TIMER	
0106	62	334	MOV A, #TIMCNT	
0107	55	335	MOV T, A	
0108	25	336	STRT T	
		337	EN TCNTI	
0109	B81E	338 ;2	CLEAR FLAGBYTE	
010B	B000	339	MOV R0, #NFLGBY	
		340	MOV @R0, #0	
010D	A5	341 ;2	FLAG1=MARK	
010E	B5	342	CLR F1	
		343	CPL F1	
		344 ;2	REPEAT	
		345	TILOP:	
		346 ;3	IF TRANSMIT REQUEST FLAG=0 THEN	
010F	B81E	347	MOV R0, #NFLGBY	
0111	F0	348	MOV A, @R0	
0112	B224	349	JBS TREC	
0114	B923	350 ;4	NXTBYTE:=PATTERN	
0116	FE	351	MOV R1, #NXTBY	
0117	A1	352	MOV A, PATT	
		353	MOV @R1, A	
0118	35	354 ;4	TRANSMIT REQUEST FLAG=1	
		355	DIS TCNTI ; LOCK OUT TIMER INTERRUPT	
		356	; SO THAT MUTUAL EXCLUSION IS MAINTAINED WHILE	
		357	; THE FLAG BYTE IS BEING MODIFIED	
0119	F0	358	MOV A, @R0	
011A	4320	359	ORL A, #TRRQFL	
011C	A0	360	MOV @R0, A	
011D	25	361	EN TCNTI	
011E	1622	362	JTF TESTA	
0120	2424	363	JMP TREC	
0122	140A	364	CALL UART ; CALL UART BECAUSE TIMER OVERFLOWED DURING LOCKOUT	
		365 ;3	ENDIF	
		366 ;3	IF DATA READY FLAG=1 THEN	
0124	F0	367	TREC: MOV A, @R0	
0125	37	368	CPL A	
0126	7238	369	JBS TREC	
		370	TREC	
		371 ;4	PATTERN:=OKDATA	
0128	B920	372	MOV R1, #OKDATA	
012A	F1	373	MOV A, @R1	
012B	AE	374	MOV PATT, A	
012C	35	375 ;4	DATA READY FLAG:=0	
		376	DIS TCNTI ; LOCK OUT TIMER INTERRUPT	
		377	; SO THAT MUTUAL EXCLUSION IS MAINTAINED WHILE	
		378	; THE FLAG BYTE IS BEING MODIFIED	
012D	F0	379	MOV A, @R0	

Figure 4 (continued)

LOC	OBJ	SEQ	SOURCE STATEMENT	THIRSTATE2	33R002	332	LOC	OBJ
012E 53F7		380	ANL A,#NOT DRDYFL					
0130 A0		381	MOV @R0,A					
0131 25		382	EN TCNTI					
0132 1636		383	JTF TESTB					
0134 2438		384	JMP TRECE					
0136 140A		385	TESTB: CALL UART ; CALL UART IF TIMER OVERFLOWED DURING LOCKOUT					
		386	TRECE:					
		387 ;3	ENDIF					
		388 ;2	UNTIL ERROR FLAG OR OVERRUN FLAG					
0138 F0		389	MOV A,@R0					
0139 5390		390	ANL A,#(OVRUN OR ERRFLG)					
013B C60F		391	JZ TILOP					
		392 ;2	INCREMENT ERROR COUNT					
013D 1F		393	INC ERRCNT					
		394 ;1	UNTIL FOREVER					
013E 2402		395	JMP TLOP					
		396 :EOF						
		397	END					
USER SYMBOLS								
ATEMP 0007	BYFNFL 0004	DRDYFL 0008	ERRCNT 0007	ERRFLG 0010	FLGBYT 0006	MARK 0080	MDATA 0021	
NFLGBY 001E	MXMTBY 0023	MOKDAT 0020	MSAMCT 001D	MTCKCT 001C	MXMTBY 0022	OMARK 0015	OSPACE 0011	
OVRUN 0080	PATT 0006	RCV000 0017	KCV010 0024	RCV020 0033	RCV030 0038	RCV040 0042	KCV045 0054	
RCV050 0059	RCV060 005F	RCV070 0061	RCVFLG 0001	REG0 0000	RETURN 0097	SAMCTK 0005	SPACE FF7F	
STPBTS 0000	SYNPLG 0002	TCKCTR 0004	TEST 0100	TESTA 0122	TESTB 0136	TILOP 010F	TIMECNT FF7E	
TISR 0007	TLOP 0102	TREC 0124	TRECE 0138	TRNGFL 0040	TRROFL 0020	UART 000A	XMI1 0064	
XMT010 006E	XMT020 007B	XMT040 0086						

ASSEMBLY COMPLETE, NO ERRORS

Figure 4 (continued)

All mnemonics copyrighted © Intel Corporation 1979.

MULTIPLY ALGORITHMS

Most microcomputer programmers have at one time or another implemented a multiply routine as part of a larger program. The usual procedure is to find an algorithm that works and modify it to work on the machine being used. There is nothing wrong with this approach. If engineers felt that they had to reinvent the wheel every time a new design is undertaken, that's probably what most of us would be doing—designing wheels. If the efficiency of the multiply algorithm, either in terms

of code size or execution time is important, however, it is necessary to be reasonably familiar with the multiplication process so that appropriate optimizations for the machine being used can be made.

To understand how multiplication operates in the binary number system, consider the multiplication of two four bit operands A and B. The "ones and zeros" in A and B represent the coefficients of two polynomials. The operation $A \times B$ can be represented as the following multiplication of polynomials:

$$\begin{array}{r}
 \begin{array}{r}
 A^3 \cdot 2^3 + A^2 \cdot 2^2 + A^1 \cdot 2^1 + A^0 \cdot 2^0 \\
 \times \quad B^3 \cdot 2^3 + B^2 \cdot 2^2 + B^1 \cdot 2^1 + B^0 \cdot 2^0
 \end{array} \\
 \hline
 \begin{array}{r}
 + B^0A^3 \cdot 2^6 + B^0A^2 \cdot 2^5 + B^0A^1 \cdot 2^4 + B^0A^0 \cdot 2^3 \\
 + B^1A^3 \cdot 2^5 + B^1A^2 \cdot 2^4 + B^1A^1 \cdot 2^3 + B^1A^0 \cdot 2^2 \\
 + B^2A^3 \cdot 2^4 + B^2A^2 \cdot 2^3 + B^2A^1 \cdot 2^2 + B^2A^0 \cdot 2^1 \\
 + B^3A^3 \cdot 2^3 + B^3A^2 \cdot 2^2 + B^3A^1 \cdot 2^1 + B^3A^0 \cdot 2^0
 \end{array}
 \end{array}$$

Figure 4 (continued)

The sum of all these terms represents the product of A and B. The simplest multiply algorithm factors the above terms as follows:

$$A \cdot B = B_0(A) \cdot 2^0 + B_1(A) \cdot 2^1 + B_2(A) \cdot 2^2 + B_3(A) \cdot 2^3$$

Since the coefficients of B (i.e., B₀, B₁, B₂, and B₃) can only take on the binary values of 1 or 0, the sum of the products can be formed by a series of simple adds and multiplications by two. The simplest implementation of this would be:

```
MULTIPLY:
  PRODUCT = 0
  IF B0 = 1 THEN PRODUCT = PRODUCT + A
  IF B1 = 1 THEN PRODUCT = PRODUCT + 2*A
  IF B2 = 1 THEN PRODUCT = PRODUCT + 4*A
  IF B3 = 1 THEN PRODUCT = PRODUCT + 8*A
END MULTIPLY
```

In order to conserve memory, the above straight line code is normally converted to the following loop:

```
MULTIPLY:
  PRODUCT = 0
  COUNT = 4
  REPEAT
    IF B[0] = 1 THEN PRODUCT = PRODUCT + A
    A = 2*A
    B = B/2
    COUNT = COUNT - 1
  UNTIL COUNT = 0
END MULTIPLY
```

The repeated multiplication of A by two (which can be performed by a simple left shift) forms the terms 2*A, 4*A, and 8*A. The variable B is divided by two (performed by a simple right shift) so that the least significant bit can always be used to determine whether the addition should be executed during each pass through the loop. It is from these shifting and addition opera-

tions that the "shift and add" algorithm takes its common name.

The "shift and add" algorithm shown above has two areas where efficiency will be lost if implemented in the manner shown. The first problem is that the addition to the partial product is double precision relative to the two operands. The other problem, which is also related to double precision operations, is that the A operand is double precision and that it must be left shifted and then the B operand must be right shifted. An examination of the "longhand" polynomial multiplication will reveal that, although the partial product is indeed double precision, each addition performed is only single precision. It would be desirable to be able to shift the partial product as it is formed so that only single precision additions are performed. This would be especially true if the partial product could be shifted into the "B" operand since one bit of the partial product is formed during each pass through the loop and (happily) one bit of the "B" operand is vacated. To do this, however, it is necessary to modify the algorithm so that both of the shifts that occur are of the same type.

To see how this can be done one can take the basic multiplication equation already presented:

$$A \cdot B = B_0(A \cdot 2^0) + B_1(A \cdot 2^1) + B_2(A \cdot 2^2) + B_3(A \cdot 2^3)$$

and factoring 2^4 from the right side:

$$A \cdot B = 2^4[B_0(A \cdot 2^{-4}) + B_1(A \cdot 2^{-3}) + B_2(A \cdot 2^{-2}) + B_3(A \cdot 2^{-1})]$$

This operation has resulted in a term (within the brackets) which can be formed by right shifts and adds and then multiplied by 2^4 to get the final result. The resulting algorithm, expanded to form an eight by eight multiplication, is shown in figure 5. Note that although the result is a full sixteen bits, the algorithm only performs eight bit additions and that only a single sixteen bit shift operation is involved. This has the effect of reducing both the code space and the execution time for the routine.

ISIS-II MCS-48/UPI-41 MACRO ASSEMBLER, V2.0

LOC	OBJ	SEQ	SOURCE STATEMENT
		1	\$MACROFILE
		2	\$INCLUDE(:F1.MPVS.HED)
		3	*****
		4	*
		5	* MPY8X8
		6	*
		7	*****
		8	*
		9	* THIS UTILITY PROVIDES AN 8 BY 8 UNSIGNED MULTIPLY
		10	* AT ENTRY:
		11	* A = LOWER EIGHT BITS OF DESTINATION OPERAND
		12	* XA= DON'T CARE
		13	* R1= POINTER TO SOURCE OPERAND (MULTIPLIER) IN INTERNAL MEMEORY

Figure 5

LOC	OBJ	SEQ	SOURCE STATEMENT
0002	BA00	= 51	1 MULTIPLICAND[15-8]:=0
0002	BD08	= 52	MOV XR, #00
		= 53	1 COUNT:=8
		= 54	MOV COUNT, #8
		= 55	1 REPEAT
		= 56	MPY8LP
0004	120E	= 57	2 IF MULTIPLICAND[0]=0 THEN BEGIN
		= 58	JB0 MPY8A
		= 59	3 MULTIPLICAND:=MULTIPICAND/2
0006	2A	= 60	XCH A, XR
0007	97	= 61	CLR C
0008	67	= 62	RRC A
0009	2A	= 63	XCH A, XR
000A	67	= 64	RRC A
000B	EB04	= 65	DJNZ COUNT, MPY8LP
000D	83	= 66	PET
		= 67	2 ELSE
		= 68	MULTIPLICAND:=MULTIPICAND+MULTIPLICAND
		= 69	ENDIF
		= 70	COUNT:=COUNT-1
		= 71	UNTIL COUNT=0
		= 72	END MPY8X8
		= 73	EQUATES
		= 74	MPY8X8=
		= 75	MPY8X8=
		= 76	MPY8X8=
		= 77	MPY8X8=
		= 78	MPY8X8=
		= 79	MPY8X8=
		= 80	MPY8X8=
		= 81	MPY8X8=
		= 82	MPY8X8=
		= 83	MPY8X8=
		= 84	MPY8X8=
		= 85	MPY8X8=
		= 86	MPY8X8=
		= 87	MPY8X8=
		= 88	MPY8X8=
		= 89	MPY8X8=
		= 90	MPY8X8=
		= 91	MPY8X8=
		= 92	MPY8X8=
		= 93	MPY8X8=
		= 94	MPY8X8=
		= 95	MPY8X8=
		= 96	MPY8X8=
		= 97	MPY8X8=
		= 98	MPY8X8=
		= 99	MPY8X8=
		= 100	MPY8X8=

Figure 5 (continued)

```

LOC OBJ SEQ SOURCE STATEMENT
= 68 MPY8A
= 69 ;3 MULTPLICAND:=MULTPLICAND*15-8;MULTIPLIER
000E 2A = 70 XCH A,XA
000F 61 = 71 ADD A,0R1
0010 67 = 72 RRC A
0011 2A = 73 XCH A,XA
0012 67 = 74 RRC A
0013 EB04 = 75 DJNZ COUNT,MPY8LP
0015 83 = 76 RET
= 77 ;3 MULTPLICAND:=MULTPLICAND/2
= 78 ;2 ENDIF
= 79 ;2 COUNT:=COUNT-1
= 80 ;1 UNTIL COUNT=0
= 81 ;1 END MPY8X8
= 82 END
USER SYMBOLS
COUNT 0003 DIGPP 0003 ICNT 0004 MPY8A 000E MPY8LP 0004 MPY8X8 0004 XA 0002
ASSEMBLY COMPLETE. NO ERRORS

```

All mnemonics copyrighted © Intel Corporation 1979.

DIVIDE ALGORITHMS

In order to understand binary division a four bit operation will again be used as an example. The following algorithm will perform a four by four division:

```

DIVIDE:
IF 16*DIVISOR>= DIVIDEND THEN
SET OVERFLOW ERROR FLAG
ELSE
IF 8*DIVISOR>= DIVIDEND THEN
QUOTIENT[3]=1
DIVIDEND:=DIVIDEND-8*DIVISOR
ELSE
QUOTIENT[3]=0
ENDIF
IF 4*DIVISOR>= DIVIDEND THEN
QUOTIENT[2]=1
DIVIDEND:=DIVIDEND-4*DIVISOR
ELSE
QUOTIENT[2]=0
ENDIF
IF 2*DIVISOR>= DIVIDEND THEN
QUOTIENT[1]=1
DIVIDEND:=DIVIDEND-2*DIVISOR
ELSE
QUOTIENT[1]=0
ENDIF
IF 1*DIVISOR>= DIVIDEND THEN
QUOTIENT[0]=1
DIVIDEND:=DIVIDEND-1*DIVISOR
ELSE
QUOTIENT[0]=0
ENDIF
ENDIF
END DIVIDE

```

The algorithm is easy to understand. The first test asks if the division will fit into the dividend sixteen times. If it will, the quotient cannot be expressed in only four bits so an overflow error flag is set and the divide algorithm ends. The algorithm then proceeds to determine if eight times the divisor fits, four times, etc. After each test it either sets or clears the appropriate quotient bit and modifies the dividend. To see this algorithm in action, consider the division of 15 by 5:

00001111	(15)	
- 01010000	(16*5)	Doesn't fit—no overflow
00001111	(15)	
- 00101000	(8*5)	Doesn't fit—Q[3]=0
00001111	(15)	
- 00010100	(4*5)	Doesn't fit—Q[2]=0
00001111	(15)	
- 00001010	(2*5)	Fits—Q[1]=1
00000101	(15-2*5)	
- 00000101	(1*5)	Fits—Q[0]=1
00000000		

The result is Q = 0011 which is the binary equivalent of 3—the correct answer. Clearly this algorithm can (and has been) converted to a loop and used to perform divisions. An examination of the procedure, however, will show that it has the same problems as the original multiply algorithm.

The first problem is that double precision operations are involved with both the comparison of the division with the dividend and the conditional subtraction. The second problem is that as the quotient bits are derived they must be shifted into a register. In order to reduce the register requirements, it would be desirable to shift them into the divisor register as they are generated since the divisor register gets shifted anyway. Unfortunately the quotient bits are derived most significant bits first so doing this will form a mirror image of the quotient—not very useful.

Both of these problems can be solved by observing that the algorithm presented for divide will still work if both sides of all the "equations" involving the dividend are divided by sixteen. The looping algorithm then would proceed as follows:

```
DIVIDE:
QUOTIENT:= 0
COUNT:= 4
DIVIDEND:= DIVIDEND/16
IF DIVISOR>= DIVIDEND THEN
OVERFLOW FLAG:= 1
ELSE
REPEAT
DIVIDEND:= DIVIDEND*2
QUOTIENT:= QUOTIENT*2
IF DIVISOR>= DIVIDEND THEN
QUOTIENT:= QUOTIENT + 1/SET QUOTIENT[0]/
DIVIDEND:= DIVIDEND - DIVISOR
ENDIF
COUNT:= COUNT - 1
UNTIL COUNT= 0
ENDIF
END DIVIDE
```

When this algorithm is implemented on a computer which does not have a direct compare instruction the comparison is done by subtraction and the inner loop of the algorithm is modified as follows:

```
REPEAT
DIVIDEND:= DIVIDEND*2
QUOTIENT:= QUOTIENT*2
DIVIDEND:= DIVIDEND - DIVISOR
IF BORROW= 0 THEN
QUOTIENT:= QUOTIENT + 1
ELSE
DIVIDEND:= DIVIDEND + DIVISOR
ENDIF
COUNT:= COUNT - 1
UNTIL COUNT= 0
```

An implementation of this algorithm using the 8049 instruction set is shown in figure 6. This routine does an unsigned divide of a 16 bit quantity by an eight bit quantity. Since the multiply algorithm of figure 5 generates a 16 bit result from the multiplication of two eight bit operands, these two routines complement each other and can be used as part of more complex computations.

IS15-II MCS-48/UPI-41 MACRO ASSEMBLER, V2.0

LOC	OBJ	SEQ	SOURCE STATEMENT
		1	\$MACROFILE
		2	\$INCLUDE(F1:DIV16. HED)
		3	*****
		4	;
		5	;
		6	;
		7	;
		8	;
		9	THIS UTILITY PROVIDES AN 16 BY 8 UNSIGNED DIVIDE
		10	AT ENTRY:
		11	A = LOWER EIGHT BITS OF DESTINATION OPERAND
		12	XA= UPPER EIGHT BITS OF DIVIDEND
		13	R1= POINTER TO DIVISOR IN INTERNAL MEMORY
		14	;
		15	AT EXIT:
		16	A = LOWER EIGHT BITS OF RESULT
		17	XA= REMAINDER

Figure 6

LOC	OBJ	SEQ	SOURCE STATEMENT	SYMBOL TABLE	LOC
		= 18 ;*	C = SET IF OVERFLOW ELSE CLEARED		*
		= 19 ;*			*
		= 20 ;*****			
		21 ;			
		22 ;			
		23 ;			
		24 ;1 DIV16			
		= 25 ;1 COUNT:=8			
		= 26 ;1 DIVIDEND(15-8):=DIVIDEND(15-8)-DIVISOR			
		= 27 ;1 IF BORROW=0 THEN /* IT FITS*/			
		= 28 ;2 SET OVERFLOW FLAG			
		= 29 ;1 ELSE			
		= 30 ;2 RESTORE DIVIDEND			
		= 31 ;2 REPEAT			
		= 32 ;3 DIVIDEND:=DIVIDEND*2			
		= 33 ;3 QUOTIENT:=QUOTIENT*2			
		= 34 ;3 DIVIDEND(15-8):=DIVIDEND(15-8)-DIVISOR			
		= 35 ;3 IF BORROW=1 THEN			
		= 36 ;4 RESTORE DIVIDEND			
		= 37 ;3 ELSE			
		= 38 ;4 QUOTIENT(0)=1			
		= 39 ;3 ENDF			
		= 40 ;3 COUNT=COUNT-1			
		= 41 ;2 UNTIL COUNT=0			
		= 42 ;2 CLEAR OVERFLOW FLAG			
		= 43 ;1 ENDF			
		= 44 ;1 ENDDIVIDE			
		45 ;			
		46 ; EQUATES			
		47 ; =====			
0002		49 ;X EQU R2			
0003		50 ;COUNT EQU R3			
		51 ;			
		52 ;EJECT			
		53 ;INCLUDE(:F1:DIV16)			
		= 54 ;1 DIV16:			
0000 2A		= 55 ;DIV16 XCH A,XA	ROUTINE WORKS MOSTLY WITH BITS 15-8		
		= 56 ;1 COUNT:=8			
0001 BB08		= 57 ;MOV COUNT,#8			
		= 58 ;1 DIVIDEND(15-8):=DIVIDEND(15-8)-DIVISOR			
0003 37		= 59 ;CPL A			
0004 61		= 60 ;ADD A,0R1			
0005 37		= 61 ;CPL A			
0006 F60B		= 62 ;1 IF BORROW=0 THEN /* IT FITS*/			
		= 63 ;JC DIV16			
		= 64 ;2 SET OVERFLOW FLAG			
0008 A7		= 65 ;CPL C			
0009 0424		= 66 ;JMP DIV16			
		= 67 ;1 ELSE			
		= 68 ;DIV16:			
		= 69 ;2 RESTORE DIVIDEND			
000B 61		= 70 ;ADD A,0R1			
		= 71 ;2 REPEAT			
		= 72 ;DIV16:			
		= 73 ;3 DIVIDEND:=DIVIDEND*2			

Figure 6 (continued)

LOC	OBJ	SEQ	SOURCE STATEMENT	THMNTAT2 30002	032	LOC	OBJ
		= 74 : 3	QUOTIENT =QUOTIENT*2				
000C	97	= 75	CLR C				
000D	2A	= 76	XCH A,XA				
000E	F7	= 77	RLC A				
000F	2A	= 78	XCH A,XA				
0010	F7	= 79	RLC A				
0011	E618	= 80	JNC DIVIE				
0013	37	= 81	CPL A				
0014	61	= 82	ADD A,001				
0015	37	= 83	CPL A				
0016	0420	= 84	JMP DIVIC				
		= 85 : 2	DIVIDEND(15-8):=DIVIDEND(15-8)-DIVISOR				
0018	37	= 86	DIVIE: CPL A				
0019	61	= 87	ADD A,001				
001A	37	= 88	CPL A				
		= 89 : 3	IF BORROW=1 THEN				
001B	E620	= 90	JNC DIVIC				
		= 91 : 4	RESTORE DIVIDEND				
001D	61	= 92	ADD A,001				
001E	0421	= 93	JMP DIVID				
		= 94 : 3	ELSE				
		= 95	DIVIC:				
		= 96 : 4	QUOTIENT(0)=1				
0020	1A	= 97	INC XA				
		= 98 : 3	ENDIF				
		= 99 : 3	COUNT =COUNT-1				
		= 100 : 2	UNTIL COUNT=0				
0021	E80C	= 101	DIVID: DJNZ COUNT,DIVILP				
		= 102 : 2	CLEAR OVERFLOW FLAG				
0023	97	= 103	CLR C				
		= 104 : 1	ENDIF				
		= 105 : 1	ENDDIVIDE				
0024	2A	= 106	DIVIS: XCH A,XA				
0025	83	= 107	RET				
		= 108	END				
USER SYMBOLS							
COUNT	0003	DIV16	0000	DIV1A	0006	DIV1B	0024
XA	0002						
ASSEMBLY COMPLETE, NO ERRORS							

Figure 6 (continued)

All mnemonics copyrighted © Intel Corporation 1979.

BINARY AND BCD CONVERSIONS

The conversion of a binary value to a BCD (binary coded decimal) number can be done with a very straightforward algorithm:

```

CONVERT_TO_BCD:
  BCDACCUM:=0
  COUNT:=PRECISION
  REPEAT
    BIN:=BIN * 2
    BCD:=BCD * 2 + CARRY
    COUNT:=COUNT - 1
  UNTIL COUNT=0
  END CONVERT_TO_BCD

```

The variable **BCDACCUM** is a BCD string used to accumulate the result; the variable **BIN** is the binary number to be converted. **PRECISION** is a constant which gives the length, in binary bits of BIN. To see how this works, assume that BIN is a sixteen bit value with the most significant bit set. On the first pass through the loop the multiplication of **BIN** will result in a carry and this carry will be added to BCD. On the remaining passes through the loop BCD will be multiplied by two 15 times. The initial carry into BCD will be multiplied by 2^{15} or 32678, which is the "value" of the most significant bit of **BIN**. The process repeats with each bit of **BIN** being introduced to **BCDACCUM** and then being scaled up on successive passes through the loop. Figure 7 shows the implementation of this algorithm for the 8049.

LOC	OBJ	SEQ	SOURCE STATEMENT
		1	\$MACROFILE
		2	\$INCLUDE('F1.CONBCD.MED')
		3	*****
		4	*
		5	CONBCD
		6	*
		7	*****
		8	*
		9	THIS UTILITY CONVERTS A 16 BIT BINARY VALUE TO BCD
		10	AT ENTRY
		11	A = LOWER EIGHT BITS OF BINARY VALUE
		12	XA = UPPER EIGHT BITS OF BINARY VALUE
		13	R0 = POINTER TO A PACKED BCD STRING
		14	*
		15	AT EXIT
		16	A = UNDEFINED
		17	XA = UNDEFINED
		18	C = SET IF OVERFLOW ELSE CLEARED
		19	*
		20	*****
		21	
		22	
		23	\$INCLUDE('F1.CONBCD.PDL')
		24	1 CONVERT_TO_BCD
		25	1 BCDACC = 0
		26	1 COUNT = 16
		27	1 REPEAT
		28	2 BIN = BIN * 2
		29	2 BCD = BCD * 2 + CARRY
		30	2 IF CARRY FROM BCDACC GOTO ERROR EXIT
		31	2 COUNT = COUNT - 1
		32	1 UNTIL COUNT = 0
		33	1 END CONVERT_TO_BCD
		34	
		35	EQUATES
		36	*****
		37	
0002		38	XA R0 EQU R2
0003		39	COUNT EQU R3
0004		40	ICNT EQU R4
		41	
0003		42	DIGPR EQU 3
		43	
		44	EJECT
		45	\$INCLUDE('F1.CONBCD')
		46	
0005		47	TEMP1 SET R5
		48	
		49	1 CONVERT_TO_BCD
		50	CNECD
		51	1 BCDACC = 0
0000 28		52	XCH R, R0

(Continued) Figure 7

The conversion of a BCD value to binary is essentially the same process as converting a binary value to BCD.

ing the following substitution:

```

CONVERT_TO_BINARY
  BIN:=0
  COUNT:=DIGNO
  REPEAT
    BCDACCUM:=BCDACCUM*10
    BIN:=10*BIN+CARRY DIGIT
    COUNT:=COUNT-1
  UNTIL COUNT=0
END CONVERT_TO_BINARY

```

$$\text{BIN} = 10 * \text{BIN} = (2) * (5) * (\text{BIN}) = 2 * (2 * 2 + 1) * \text{BIN}$$

The only complexity is the two multiplications by ten. The BCDACCUM can be multiplied by ten by shifting it left four places (one digit). The variable BIN could be multiplied using the multiply algorithm already discussed, but it is usually more efficient to do this by mak-

This implies that the value $10 * \text{BIN}$ can be generated by saving the value of BIN and then shifting BIN two places left. After this the original value of BIN can be added to the new value of BIN (forming $5 * \text{BIN}$) and then BIN can be multiplied by two. It is often possible to implement the multiplication of a value by a constant by using such techniques. Figure 8 shows an 8049 routine which converts BCD values to binary. This routine differs slightly from the algorithm above in that the BCD digits are read, and converted to binary, two digits at a time. Protection has also been added to detect BCD operands which, if converted, would yield binary values beyond the range of the result.

IS15-II MCS-48/UPI-41 MACRO ASSEMBLER, V2.0

```

LOC OBJ      SEQ      SOURCE STATEMENT
1 $MACROFILE
2 $INCLUDE(< F1:CONBIN.HED>)
3 :*****
4 :*
5 :*      CONBIN
6 :*
7 :*****
8 :*
9 :*      THIS UTILITY CONVERTS A 6 DIGIT BCD VALUE TO BINARY
10 :*      AT ENTRY:
11 :*      R0= POINTER TO A PACKED BCD STRING
12 :*
13 :*      AT EXIT:
14 :*      A = LOWER EIGHT BITS OF THE BINARY RESULT
15 :*      XA= UPPER EIGHT BITS OF THE BINARY RESULT
16 :*      C = SET IF OVERFLOW ELSE CLEARED
17 :*
18 :*****
19 :
20 :
21 $INCLUDE(< F1:CONBIN.PDL>)
22 :
23 :
24 :1 CONVERT_TO_BINARY
25 :1 POINTER0:=POINTER0+DIGITPAIR-1
26 :1 COUNT:=DIGITPAIR
27 :1 BIN:=0
28 :1 REPEAT
29 :2   BIN:=BIN*10
30 :2   BIN:=BIN+MEM(R0)[7-4]
31 :2   BIN:=BIN*10
32 :2   BIN:=BIN+MEM(R0)[3-0]

```

LOC	OBJ	SEQ	SOURCE STATEMENT
		= 33 ; 2	POINTER0 = POINTER0 - 1
		= 34 ; 2	COUNT = COUNT - 1
		= 35 ; 1	UNTIL COUNT = 0
		= 36 ; 1	END CONVERT_TO_BINARY
		37 ;	
		38 ;	EQUATES
		39 ;	=====
		40 ;	
0002	41	XA	EQU R2
0003	42	COUNT	EQU R3
0004	43	ICNT	EQU R4
	44 ;		
0003	45	DIGPR	EQU 3
	46 ;		
	47 ;		#EJECT
		48	\$INCLUDE(F1:CONBIN)
		= 49 ;	
0005	= 50	TEMP1	SET R5
0006	= 51	TEMP2	SET R6
	= 52 ;		
	= 53 ; 1		CONVERT_TO_BINARY
	= 54	CONBIN:	
	= 55 ; 1	POINTER0 = POINTER0 + DIGITPAIR - 1	
0000 F8	= 56	MOV	A, R0
0001 0302	= 57	ADD	A, #DIGPR - 1
0003 A8	= 58	MOV	R0, A
	= 59 ; 1	COUNT = DIGITPAIR	
0004 B003	= 60	MOV	COUNT, #DIGPR
	= 61 ; 1	BIN = 0	
0006 27	= 62	CLR	A
0007 AA	= 63	MOV	XA, A
	= 64 ; 1	REPEAT	
	= 65	CONBLP:	
	= 66 ; 2	BIN = BIN * 10	
0008 142B	= 67	CALL	CONB10
000A F62A	= 68	JC	CONBER
	= 69 ; 2	BIN = BIN + MEM(R0)[7-4]	
000C AD	= 70	MOV	TEMP1, A
000D F0	= 71	MOV	A, R0
000E 47	= 72	SWAP	A
000F 530F	= 73	ANL	A, #0FH
0011 6D	= 74	ADD	A, TEMP1
0012 2A	= 75	XCH	A, XA
0013 1300	= 76	ADDC	A, #00
0015 2A	= 77	XCH	A, XA
0016 F62A	= 78	JC	CONBER
	= 79 ; 2	BIN = BIN * 10	
0018 142B	= 80	CALL	CONB10
001A F62A	= 81	JC	CONBER
	= 82 ; 2	BIN = BIN + MEM(R0)[3-0]	
001C AD	= 83	MOV	TEMP1, A
001D F0	= 84	MOV	A, R0
001E 530F	= 85	ANL	A, #0FH
0020 6D	= 86	ADD	A, TEMP1
0021 2A	= 87	XCH	A, XA

LOC	OBJ	SEQ	SOURCE STATEMENT
0022	1300	= 88	ADDC A, #00
0024	2A	= 89	XCH A, XA
0025	F62A	= 90	JC CONBER
		= 91 ; 2	POINTER0:=POINTER0-1
0027	C8	= 92	DEC R0
		= 93 ; 2	COUNT:=COUNT-1
		= 94 ; 1	UNTIL COUNT=0
0028	EB08	= 95	DJNZ COUNT, CONBLP
		= 96 ; 1	END CONVERT_TO_BINARY
002A	83	= 97	CONBER: RET
		= 98	#EJECT
		= 99 ;	
		= 100 ;	
		= 101 ;	UTILITY TO MULTIPLY BIN BY 10
		= 102 ;	CARRY WILL BE SET IF OVERFLOW OCCURS
		= 103 ;	
002B	AD	= 104	CONB10: MOV TEMP1, A ; SAVE A
002C	2A	= 105	XCH A, XA ; SAVE XA
002D	AE	= 106	MOV TEMP2, A
002E	2A	= 107	XCH A, XA
		= 108 ;	
002F	97	= 109	CLR C
0030	F7	= 110	PLC A ; BIN:=BIN*2
0031	2A	= 111	XCH A, XA
0032	F7	= 112	RLC A
0033	2A	= 113	XCH A, XA
0034	F646	= 114	JC CONB1E ; ERROR ON OVERFLOW
		= 115 ;	
0036	F7	= 116	RLC A ; BIN:=BIN*4
0037	2A	= 117	XCH A, XA
0038	F7	= 118	RLC A
0039	2A	= 119	XCH A, XA
003A	F646	= 120	JC CONB1E ; ERROR ON OVERFLOW
		= 121 ;	
003C	6D	= 122	ADD A, TEMP1 ; BIN:=BIN*5
003D	2A	= 123	XCH A, XA
003E	7E	= 124	ADDC A, TEMP2
003F	2A	= 125	XCH A, XA
0040	F646	= 126	JC CONB1E ; ERROR ON OVERFLOW
		= 127 ;	
0042	F7	= 128	RLC A ; BIN:=BIN*10
0043	2A	= 129	XCH A, XA
0044	F7	= 130	RLC A
0045	2A	= 131	XCH A, XA
		= 132 ;	
0046	83	= 133	CONB1E: RET
		= 134	
		= 135 ;	
		= 136	END

USER SYMBOLS

CONB10	002B	CONB1E	0046	CONBER	002A	CONBIN	0000	CONBLP	0008	COUNT	0003	DIGPR	0003	ICNT	0004
TEMP1	0005	TEMP2	0006	XA	0002										

ASSEMBLY COMPLETE, NO ERRORS

CONCLUSION

The design goals of the full duplex serial communications software were realized; if transmission and reception are occurring concurrently, only 42 percent of the real time available to the 8049 will be consumed by the serial link. This implies that an 8049 running full duplex serial I/O will still outperform earlier members of the family running without the serial I/O requirement. It is also possible to run this program in an 8048 or 8748 at 1200 baud with the same 42 percent CPU utilization.

The execution times for the other routines that have been discussed have been summarized in Table 1. All of these routines were written to maintain maximum usability rather than minimum code size or execution time. The resulting execution times and code size are therefore what the user can expect to see in a real application. The results that were obtained clearly show the efficiency and speed of the 8049. The equivalent times for the 8048 are also shown. It is clear that the 8049 represents a substantial performance advantage over the 8048. Considering, in most applications, that the 8048 is

the highest performance microcomputer available to date, the performance advantage of the 8049 should allow the cost benefits of a single chip microcomputer to be realized in many applications which up until now have required too much "computer power" for a single chip approach.

	EXECUTION TIME (MICROSECONDS)		
	BYTES	8049	8048
MPY8	21	109	200
DIV 16	37	183 MIN 204 MAX	335 MIN 375 MAX
CONBCD	36	733	1348
CONBIN	70	388	713

Table 1. Program Performance

NOTE

- * Any number of breakpoints may be used in any combination;
- * "Auto-Step" operation causes the current program counter and Accumulator contents to be printed on the display for a short time on every instruction cycle;
- * "Auto-Break" provides the above display only when a break flag is encountered, with real time operation otherwise;
- * While running in non-break mode, a TTL level output is generated whenever a break flag is encountered. This signal may be used to trigger an oscilloscope or logic Analyzer to assist in hardware and software debugging.

While debugging, any instruction which causes a display are "alive". Execution may be suspended or terminated by commands from the keyboard.

Intent of this Note

While the HSE-48 emulator can assist a new microcomputer user in becoming familiar with the 8048 and 8049 microcomputers, its inherent debug capabilities will also prove helpful to design engineers. The circuit could be used for new system development and verification or adapted for prototyping production.

The main concern in designing the HSE-48 emulator was to keep the basic design simple, while maximizing the system's flexibility. The design allows the user to allow jumpers, hardware and software switches, etc. to allow the user to reconfigure the system according to the way he dedicates chip-select pins, I/O, etc. The emulator can be changed to fit each user's unique needs, rather than forcing the user to alter his needs to what is provided.

The primary intent of note is to provide the reader with the information needed to reconstruct and make full use of the HSE-48 emulator. Less emphasis is placed on describing how the hardware operates or how the commands are implemented. This information may be found in the schematic diagrams and software listings included in the Appendices.

III. GENERAL HARDWARE OVERVIEW

User Program Emulation

The actual emulation of the user's program is done using an 8039 microcomputer (IC2) on the system board. The basic minimum configuration includes an 8039 microcomputer, an 8255 address latch (IC1), and 256K RAM to use for program development and real-time execution (ICs B1, C1, and C2). Address and data RAMs are added to allow the user to expand program and data memory to 4K each. The 8039 microcomputer is full time until RAMs must be used.

Application Note presents a description of the design and operation of a high-speed emulator for the MCS-48 family of single chip microcomputers. The HSE-48 emulator provides a simple and inexpensive means for developing and debugging 8048 programs while requiring only 11-MHz operating speed of the 8039.

Section II of this Application Note describes some of the features of the development tool and how it may be used. Section III briefly discusses the hardware used to implement the emulator, while Section IV describes the manner in which program execution status is made available to the user.

A detailed description of all of the operator commands is presented in Section V of this note, along with the tables in which may be specified for each command.

August 1979

The HSE-48 system is detailed and explained in Section VI. Section VII shows how the basic circuit may be modified to provide options on memory organization, I/O configurations, etc.

The schematic of the system hardware, as well as the software programs, are presented in Appendices A and B, respectively. A short summary of the command syntax is presented in Appendix C. Appendix D explains the error message codes which may appear during operation.

The segment of the reader is already familiar with the operation of the 8048 or 8049 microcomputers. Some knowledge of the MCS-48 architecture is needed to understand sections of the command and modifier descriptions. Most users will already have this background. Further information is referred to the MCS-48 Microcomputer User's Manual, publication number 9800370.

THE HSE-48 EMULATOR TOOL

In response, the HSE-48 emulator provides the user a means to execute a program located in external memory, such as EPROM. This is done by downloading the program to the emulator's program memory, which is done either manually or via a serial link. Once the program is downloaded, the user can execute the program and observe the results on the display. The emulator also provides a means to download the program back to the external memory. The following modes are provided: Program in full (11 MHz), Program in full (11 MHz) with breakpoints, Program in full (11 MHz) with breakpoints and single step, Program in full (11 MHz) with breakpoints and single step and watch.

Applications Staff
Microcontroller Operation

I. PURPOSE AND SCOPE

This Application Note presents a description of the design and operation of a high-speed emulator for the Intel® MCS-48™ family of single chip microcomputers. The HSE-49™ emulator provides a simple and inexpensive means for executing and debugging 8049 programs which require the full 11-MHz operating speed of the part.

Section II of this Application Note describes some of the features of this development tool and how it may be used. Section III briefly discusses the hardware used to implement these features, while Section IV describes the manner in which program execution status is made available to the operator.

A detailed description of all of the operator commands is presented in Section V of this note, along with the modifiers and options which may be specified for each command. Known restrictions and limitations of the HSE-49 system are listed and explained in Section VI. Section VII shows how the basic circuit may be modified to provide options on memory organization, I/O configurations, etc.

Full schematics of the system hardware, as well as monitor software listings, are presented in Appendices A and B, respectively. A short summary of the command syntax is presented in Appendix C. Appendix D explains the error message codes which may appear during use.

It is assumed that the reader is already familiar with the operation of the 8048 or 8049 microcomputers. Some knowledge of the 8048 architecture is needed to understand sections of the command and modifier descriptions. Most users will already have this background. Other readers are referred to the *MCS-48 Microcomputer User's Manual*, Intel publication number 9800270.

II. THE HSE-49 DEVELOPMENT TOOL

In essence, the HSE-49 emulator provides the user a means for executing an MCS-48 program located in external RAM rather than internal ROM or EPROM. This allows programs being debugged to be modified easily and quickly during the debug cycle. A user's program may be entered into system RAM either manually or via a serial link from a host computer such as an Inteltec® Microcomputer Development System. Once loaded, the program can be modified using an on-board keyboard and display, and executed in real-time in a number of breakpoint modes. The internal state of the processor, including RAM, accumulator, timer/counter, and status register contents, can also be read and modified through the keyboard.

Breakpoint and debug facilities are extremely flexible. The following execution modes are provided.

- Programs may be run in full (11 MHz) real time;
- Programs may be single-stepped;
- In break mode, programs run in full real time until break occurs;

- Breaks may be triggered by either program or external data RAM accesses;
- Any number of breakpoints may be used in any combination;
- "Auto-Step" operation causes the current program counter and Accumulator contents to be printed on the display for a short time on every instruction cycle;
- "Auto-Break" provides the above display only when a break flag is encountered, with real time operation otherwise;
- While running in non-break mode, a TTL-level pulse is generated whenever a break flag is encountered. This signal may be used to trigger an oscilloscope or Logic Analyzer to assist in hardware and software debug.
- While running in any mode, the keyboard and display are "alive". Execution may be suspended or terminated by commands from the keyboard.

Intent of this Note

While the HSE-49 emulator can assist a new microcomputer user in becoming familiar with the 8048 and 8049 microcomputers, its inherent debug capabilities will also prove helpful to design engineers. The design could be used for new system development and verification or adapted for prototype production.

The main concern in designing the HSE-49 emulator was to keep the basic design simple, while maximizing the system's flexibility. The design allows the use of jumpers, hardware and software switches, etc. to allow the user to reconfigure the system according to the way he dedicates chip-select pins, I/O, etc. The emulator can be changed to fit each user's unique needs, rather than forcing the user to alter his needs to what is provided.

The primary intent of note is to provide the reader with the information needed to reconstruct and make full use of the HSE-49 emulator. Less emphasis is placed on describing how the hardware operates or how the commands are implemented. This information may be found in the schematic diagrams and software listings included in the Appendices.

III. GENERAL HARDWARE OVERVIEW

User Program Emulation

The actual emulation of the user's program is done using an 8039 microcomputer (IC29 on the schematics in Appendix A) executing a program stored in external RAM. The basic minimum configuration includes the 8039 microcomputer, an 8282 address latch (IC19), and 2K bytes of 2114 RAM to use for program development and real-time execution (ICs B1, C1, B2, and C2). Additional RAM may be added to allow the user to expand his program and data memory to 4K each. (If an 11-MHz crystal is used with the microcomputer, type 2114-3 RAMs must be used.)

System Supervision

A second microcomputer — another 8039 (IC25) with an 8282 address latch (IC16) and off-chip program memory in a 2716 EPROM (IC15) — is used to scan the on-board keyboard and display, interpret and implement commands, drive serial interfaces, etc. In general, the master processor is used to interface the execution processor's memory spaces with the outside world and control the operation of the execution processor. In this note the two processors will be abbreviated "MP" and "EP", respectively. Figure 1 shows how the two processors interrelate with the rest of the system.

Keyboard/Display

The 33-key keyboard shown in Figure 2 includes a 16-key hexadecimal keypad and 17 special function keys for specifying commands and modifiers. Readers already

familiar with the PROMPT-48™ debug tool for the 8048 will find that 25 of the HSE-49 emulator keys are identical in function and layout to the PROMPT-48 keyboard, and use the PROMPT-48 command syntax. The eight additional keys are used to generalize and augment the PROMPT-48 capabilities, as described in Section V.

The eight-character seven-segment display (DS1-DS8) is used for displaying addresses, data, and pseudo-alphanumeric messages. The display responses printed in Section V and throughout this note use a mix of upper and lower case letters to indicate what seven-segment patterns appear. An 8243 (IC9) and eight DIP packages (resistor packs, current buffers, etc.) are used for multiplexing the display and scanning the keyboard.

Breakpoint Detection

Breakpoints are specified and detected using a 2102A 1K x 8 RAM corresponding to each pair of 2114s (ICs A1

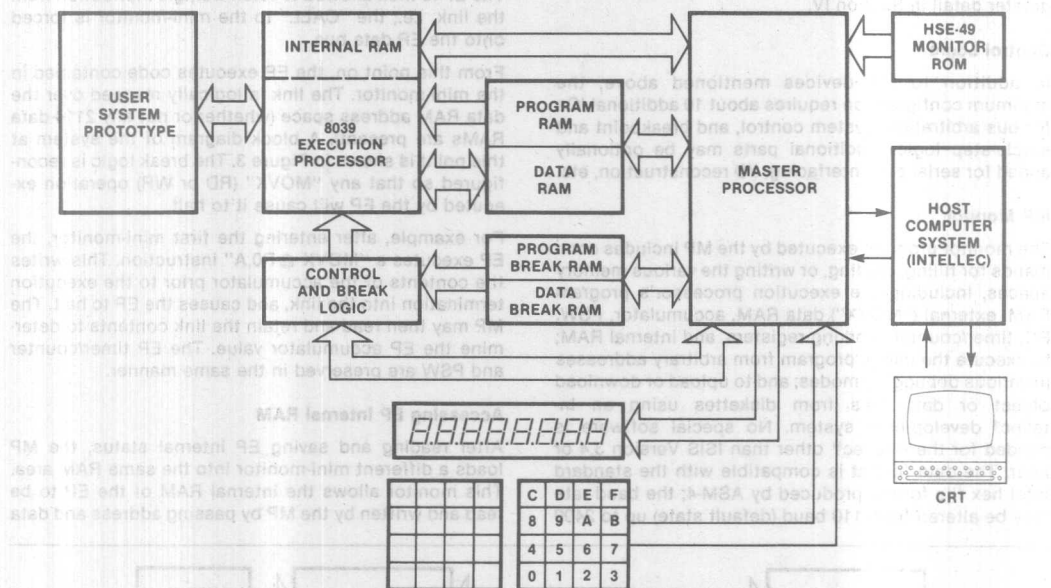


Figure 1. HSE-49™ Emulator Signal Flow Diagram



Figure 2. HSE-49™ Emulator Command Keyboard Organization

and A2). In effect, each program or data address accesses a 9-bit word. Eight bits are used normally for code or data storage. The ninth bit, accessed in parallel with the other eight, is used to indicate if a breakpoint has been set for that address. This output, when asserted, is latched (IC27 and IC36) and used to halt the execution processor via the single-step input. (In other modes, the break logic can be reconfigured to set the break requested flip-flop on any EP machine cycle or any EP "MOVX" instruction.)

Link Register

An 8212 8-bit latch (IC18) is used to communicate data and commands between the master and control processors. Under control of the MP, this register, called the "Link" register, may be logically mapped into either the program or data RAM address spaces. When this is done, the 2114s in the respective memory space are disabled and the link responds to all accesses, regardless of address. The link will be discussed in greater detail in Section IV.

Control Logic

In addition to the devices mentioned above, the minimum configuration requires about 10 additional ICs for bus arbitration, system control, and breakpoint and single-step logic. Additional parts may be optionally added for serial port interfacing, I/O reconstruction, etc.

MP Monitor

The monitor program executed by the MP includes commands for filling, reading, or writing the various memory spaces, including the execution processor's program RAM, external ("MOVX") data RAM, accumulator, PSW, PC, timer/counter, working registers, and internal RAM; to execute the user's program from arbitrary addresses in various debugging modes; and to upload or download object or data files from diskettes using an Inteltec® development system. No special software is needed for the Inteltec® other than ISIS Version 3.4 or later. The data format is compatible with the standard Intel hex file format produced by ASM-4; the baud rate may be altered from 110 baud (default state) up to 2400

baud from the on-board keyboard. Blocks of data may be transmitted to a CRT or printer and displayed in a tabular format.

IV. INTERPROCESSOR COMMUNICATION

Program Break Sequence

When the MP detects that the EP has been halted by the breakpoint hardware, or when the operator presses a key while the program is executing, the program break sequence is initiated. The low-order 23 bytes of user program memory is read into a buffer within the internal RAM of the MP. A short program for reading and transmitting internal EP status is written over the low-order program memory. (This is one of several "mini-monitors" overlayed over the user program area.) The link register is mapped logically over the user program memory, and loaded with the 8049 machine code for a "CALL" instruction to the mini-monitor program area. The EP is then allowed to fetch a single instruction from the link, i.e., the "CALL" to the mini-monitor is forced onto the EP data bus.

From this point on, the EP executes code contained in the mini-monitor. The link is logically mapped over the data RAM address space (whether or not any 2114 data RAMs are present). A block diagram of the system at this point is shown in Figure 3. The break logic is reconfigured so that any "MOVX" (RD or WR) operation executed by the EP will cause it to halt.

For example, after entering the first mini-monitor, the EP executes a "MOVX @R0,A" instruction. This writes the contents of the accumulator prior to the execution termination into the link, and causes the EP to halt. The MP may then read and retain the link contents to determine the EP accumulator value. The EP timer/counter and PSW are preserved in the same manner.

Accessing EP Internal RAM

After reading and saving EP internal status, the MP loads a different mini-monitor into the same RAM area. This monitor allows the internal RAM of the EP to be read and written by the MP by passing address and data

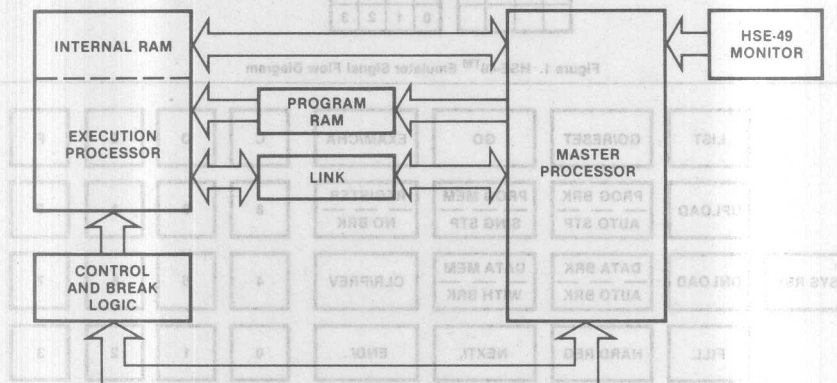


Figure 3. Communication between EP & MP

values between the two processors using the link register.

This is needed for two reasons. First, the EP program counter prior to the forced "CALL" instruction may be derived from the EP stack contents, and may be modified to cause the EP to resume execution at any desired address. Secondly, the internal RAM of the EP may then be accessed and modified in the process of executing a number of the monitor commands.

Resuming User Program Execution

In order to resume user program execution, a status-restoration mini-monitor is overlayed. This restores the EP internal status using a scheme analogous to the one in which the status was originally saved. The final step of the last mini-monitor is an "RETR" instruction, after which the EP is again halted. The low-order program memory saved earlier is rewritten into the appropriate area, the break logic is reconfigured for the desired execution mode, and the EP is released to run at full speed until the next break situation is encountered.

Note that all commands are implemented using "logical" rather than "physical" addressing. Thus the operator need not be concerned with the intricacies of the system design. For example, when any monitor command refers to low-order user program memory, the appropriate byte of storage within the MP internal RAM is accessed instead. If the location is altered, the internal RAM is modified appropriately. When program memory is reloaded prior to resuming user program execution, the modified version of the user program will be the one loaded.

Baud	HR06	HR07
110	93H	04H
150	96H	03H
300	45H	02H
600	9DH	01H
1200	44H	01H
2400	1AH	01H

Table 1. Serial Interface Data Rate Parameters

V. HSE-49 COMMAND DESCRIPTION

Whenever the characters "HSE-49" are present on the system display, a command string may be entered by the operator. In general, all command strings consist of a basic command initiator, an optional command modifier or type-designator, and a number of parameters or delimiters entered as hexadecimal digits. A command is executed, or a command in progress terminated, by pressing the [END/.] key. Logical default values are assumed for the modifier and parameters if either (or both) are omitted. A default parameter assumed for the command modifier will be presented on the display when the first parameter is entered.

Each parameter is a string of up to three hexadecimal digits. If more than three digits are entered, only the most recent three are considered. This allows an erroneous digit to be corrected without respecifying the entire command. A parameter is completed by pressing the [NEXT/] key. Some commands may only need the

low order part of a parameter; i.e., a command incorporating a data byte (such as [FILL]) will use only the low-order 8 bits of the corresponding parameter; internal RAM and hardware register addressing uses only seven. In each case, higher order bits are ignored.

A command string is terminated and the command invoked by pressing the [END/.] key. The command will also be invoked by pressing the [NEXT/] key when no additional parameters are allowed. A command string may be aborted at any point before the command is invoked by pressing the [CLEAR/PREV] key, and the sign-on message will appear.

Errors

An illegal command string, command terminator, or hardware failure will cause an error message and error code number to appear on the display (e.g., "Error-3"). When this occurs, the monitor can be returned to command mode by pressing the [CLEAR] or [END/.] keys. An explanation of the various error codes is given in Appendix D.

Command Classes

Commands for the HSE-49 emulator are divided into general classes, where all commands in each class have the same choice of options or modifiers. A brief description of each command, followed by a description of the allowed options, is presented below by class.

Data Manipulation/Control Command Group

Commands:

[EXAM/CHA]

Display Response — "ECh."

Function — Examine/change memory location.

Causes the memory address specified to be read and presented on the display. New data may be entered (if desired) from the hexadecimal keypad. New data is verified before appearing on the display. Subsequent or previous locations may be read by pressing the [NEXT/] or [PREV] keys, respectively. Command terminated with [END/.] key.

[FILL]

Display Response — "FIL."

Function — Fill range of memory addresses with a single data value.

Fill the appropriate memory space between the addresses specified by the first two parameters with the low-order byte of the third parameter. If second parameter less than first, only the location specified by the first is affected. If third parameter omitted, zero is assumed. If second and third parameters omitted, individual address specified is cleared. Command is useful for setting a large range of breakpoints; e.g., all of page 3 may be enabled for break with the command:

[FILL][PROG BRK]<300>[,<3FF>][,<1>]

[LIST]

Display Response — "Lst."

Function — List memory to output device through HSE-49 serial port.

Display the contents of a range of addresses given by two parameters to a teletype or CRT screen. Data is formatted, 16 separated bytes per line, with the starting address of each line printed. If used with an Intellec® system, the operator first uses ISIS-II to transfer the TTY input to the CRT output ("COPY :TI: TO :CO:") then invokes this command from the keypad. Alternatively, any ISIS device or disk file name(:TO:, :LP:, :F1:HRDREG.SAV, etc.) may be used as the destination.

[DNLOAD]

Display Response — "dnL."

Function — Download memory through HSE-49 serial port

Load data in hex file format through the serial input port. If used with Intellec® system, the operator first invokes this command from the keypad, then uses ISIS-II to transfer a disk file to the teletype port ("COPY :Fn:file.HEX TO :TO:").

The use of the checksum field for the download command is expanded slightly over the Intel hex file format standard. If the first character of the checksum field is a question mark ("?"), the checksum for that record will not be verified. This allows large object files produced by the assembler to be patched using the ISIS text editor without the necessity of manually recomputing the checksum value.

[UPLOAD]

Display Response — "UPL."

Function — Upload memory through HSE-49 serial port.

Output the contents of a range of addresses specified by the two parameters through the HSE-49 serial port in standard Intel hex file format. If used with Intellec® system, the operator first uses ISIS-II to transfer the TTY input to a disk file ("COPY :TI: TO :Fn:file.HEX"), then invokes this command from the keypad.

Data types allowed:

[PROG MEM]

Display Response — "Pr."

Function — User program memory.

Memory used to develop and execute user program. Addresses 000 through 7FF are the execution processor's memory bank 0; 800 through FFF are memory bank 1.

[REGISTER]

Display Response — "rG."

Function — Register memory and RAM.

Internal RAM of execution processor. Locations 0-7 are working register bank 0; 18-1F are working register bank 1. Only the low-order 7 bits of an address are considered.

[DATA MEM]

Display Response — "dA."

Function — External data memory (if installed).

Memory accessed by execution processor "MOVX A,@Rr" or "MOVX @Rr,A" instructions. High-order 4 bits may or may not be relevant, depending on jumpering option selected (explained in Section VII of this note).

[HARD REG]

Display Response — "Hr."

Function — Hardware registers.

The execution processor (EP) hardware registers (accumulator, timer/counter, etc.), as well as several parameters for controlling HSE-49 system status, are accessible through this catch-all memory space. Addresses are as follows:

00 — EP accumulator.

01 — EP PSW.

Bits correspond to 8049 PSW except that bit 3 (unused in the 8049) is used to monitor and alter the state of F1. Bits 2-0 correspond to the stack pointer value after the EP executes a CALL to the mini-monitor; i.e., one greater than when EP was running the user's program.

02 — EP timer/counter.

03 — EP internal RAM location 00.

(This value is also accessible through [REGISTER] space.)

04 — EP program counter (low byte).

05 — EP program counter (high nibble).

06-07 — HSE-49 serial interface baud rate parameters. Defaults to 110 baud; other rates may be selected by loading the values listed in Table 1.

08 — HSE-49 automatic sequencing rate parameter. Used in [GO][AUTO STP] and [GO][AUTO BRK] execution commands. 00 → fastest; FF → slowest. Defaults to 20H; approximately two steps per second.

09 — Monitor version/release number (packed BCD).

0A-0F — Currently unused by the monitor program.

10-7F — Variables used by master processor (MP) monitor. Should not be altered by operator.

[PROG BRK]

Display Response — "Pb."

Function — User program breakpoint memory.

Memory space used to indicate points where program execution should halt when running in a mode with breakpoints enabled ([GO][W/ BRK] and [GO][AUTOBRK]). Break will occur if enabled byte is read as the first or last byte of a 2-byte instruction, or read in executing a MOVP, MOVP3, or JMPP instruction. Memory is only 1 bit per location; 00 indicates continue, 01 causes a halt. Addresses 000 through 7FF are the execution processor's memory bank 0; 800 through FFF are memory bank 1.

[DATA BRK]

Display Response — "db."

Function — External data RAM breakpoint memory.

Memory space used to indicate points where data accesses should halt when running in a mode with breakpoints enabled ([GO][W/ BRK] and [GO][AUTOBRK]). Memory is only 1 bit per location; 00 indicates continue, 01 causes a halt. High-order 4 bits of breakpoint address may or may not be relevant, dependent on jumpering option selected for the corresponding data RAM (explained in Section VII of this note).

User Program Execution Control Group

Commands:

[GO]

Display Response — "Go."

Function — Begin execution.

If a parameter is given as part of the command string, execution will begin at that address. Otherwise, the EP program counter (hardware registers 04 and 05) will be used. These will contain the program counter from an earlier program execution break unless they have since been explicitly modified by the operator.

If command is terminated by [END/], the EP's F1, PSW and stack pointer will be cleared. If command string is terminated by [NEXT/], PSW will be taken from the EP PSW contents (hardware register 01).

While running the user's program, the characters "--run--" are written on the display. Execution may be halted and another command initiated by pressing the appropriate command key. Execution may be suspended at any time in any mode by pressing the [END/] key. This will cause the current value of the execution processor program counter and accumulator to appear on the display in the form "PC.234-56". System status is saved in the appropriate hardware registers. At this point, or when an enabled breakpoint is encountered, pressing the [NEXT/] key will cause the program to continue in the same mode as before. Any other command may be invoked by pressing the appropriate command string.

[GO/RESET]

Display Response — "Gr."

All mnemonics copyrighted © Intel Corporation 1976.

Function — Go from reset state.

EP is hardware-reset and released to execute the user's program from location 000H. No parameters are allowed. F0, F1, PSW, stack pointer, memory bank flip-flop, etc. are cleared.

Note that this command does not require the use of mini-monitors to initiate program execution. As the last phase of the program development cycle, the 2114 program RAMs and address decoder may be removed and replaced by a ROM or EPROM part (not shown in schematics). This command may be used to start execution when the program RAM has been removed. No interrogation of EP status or internal RAM may be done, nor are break or single-step modes allowed in this case, though the 2102A breakpoint RAM outputs may still be used to trigger a logic analyzer.

Execution modes allowed:

[NO BRK]

Display Response — "nb."

Function — Without breakpoints.

Full-speed execution without breakpoints enabled. Does not affect the state of the breakpoint memories.

[SING STP]

Display Response — "SSt."

Function — Single Step.

Step through program one instruction at a time. After each instruction is executed, execution halts with the current value of the Execution Processor Program Counter and Accumulator appearing on the display in the form "PC.234-56". System status is saved in the appropriate Hardware Registers. At the point, [NEXT/] will cause the program to execute one more instruction, or any other command may be invoked by pressing the appropriate command string. Does not affect the state of the Breakpoint Memories.

[W/ BRK]

Display Response — "br."

Function — With breakpoints.

Full-speed execution with breakpoints enabled. When a breakpoint is encountered, execution halts with the current value of the execution processor program counter and accumulator appearing on the display in the form "PC.234-56". System status is saved in the appropriate hardware registers. At this point, [NEXT/] will cause the program to continue until the next breakpoint is reached, or any other command may be invoked by pressing the appropriate command string.

[AUTO STP]

Display Response — "ASt."

Function — Automatically sequence through a series of instructions.

Step through program one instruction at a time. After each instruction is executed, execution halts with the current value of the execution processor program counter and accumulator appearing on the display in the form "PC.234-56". System status is saved in the appropriate hardware registers. Execution resumes after a time determined by contents of hardware register 08. Does not affect the state of the breakpoint memories.

[AUTO BRK]

Display Response — "Abr."

Function — Automatically sequence between breakpoints.

Execute a series of instructions in real time between breakpoints. When breakpoint is encountered, halt EP temporarily while program counter and accumulator contents are displayed, then continue. Display is sustained after execution resumes. Does not affect the state of the breakpoint memories.

Breakpoint Control Command Group

Commands:

[B]

Display Response — "Stb."

Function — Breakpoint set.

Set breakpoint for the address given. Multiple breakpoints may be set by entering additional addresses, separated by the [NEXT/] key. Command terminated by pressing [END/]. Action taken is to fill the appropriate breakpoint memory locations with logical ones.

[C]

Display Response — "CLb."

Function — Clear breakpoint.

Clear breakpoint for the address given. Multiple breakpoints may be cleared by entering additional addresses, separated by the [NEXT/] key. Command terminated by pressing [END/]. Action taken is to fill the appropriate breakpoint memory locations with logical zeroes.

Data types allowed:

[PROG MEM]

Display Response — "Pr."

Function — Break on program memory fetch.

Applies command to the program breakpoint memory space.

[DATA MEM]

Display Response — "dA."

Function — Break on data memory access.

Applies command to the external data breakpoint memory space.

System Control Command Group

Command:

[SYS RST]

Display Response — "HSE-49."

Function — System reset.

Reset both the MP and EP and clear all breakpoints (requires approximately one second). CAUTION — If reset while EP is executing the user's program, the low order section of program memory (about 23 bytes) will be altered.

VI. SYSTEM LIMITATIONS

In designing the HSE-49 emulator, certain compromises were made in an attempt to maximize the usefulness of the emulator while keeping the circuitry simple and inexpensive. As a result, the following limitations exist and must be taken into account when using the system.

1. As explained in Section IV, user program execution is terminated (by single-stepping, breakpoints, pressing the [END/] key, etc.) by forcing the execution processor to execute a "CALL" instruction to the mini-monitor. This uses one level of the EP subroutine stack. The EP PSW reflects the value of the stack pointer *after* processing this CALL. As a result, the value indicated for stack depth by examining the EP PSW (hardware register 01) is one greater than the depth when the break was initiated. The user program must not be using all eight levels of stack when a break is initiated or the bottom level will be destroyed.
2. User program is initiated (by the [GO] command or when resuming execution after a breakpoint, single-stepping, etc.) by forcing the EP to execute an "RETR" instruction. This will clear the EP interrupt-in-progress flip-flop. If the user program allows both external and timer interrupts to be enabled at the same time, care must be taken to avoid causing a break while the EP is within an interrupt servicing routine. No limitation is placed on breakpoints or single-stepping in the background program because of this.
3. When the user program execution is terminated (by a break, single-stepping, etc.) and later resumed, the EP timer/counter is restored to its value when the break occurred (unless modified by the user). The prescaler, however, will have changed. Thus, up to 31 machine cycles may be "lost" or "gained" if a break occurs while the timer is running.
4. Timer interrupts occurring at the same time as an EP break may be ignored if the timer overflow occurs after breaking user program execution before the timer value is saved.
5. The 8049 "RET" and "RETR" instructions are each 1-byte, 2-cycle instructions. During the second cycle the byte following the return instruction is fetched and ignored. If a program breakpoint is set for a location following a "RET" or "RETR" instruction, a break will be initiated when the return is executed.

6. Breakpoints should not be placed in the last 3 bytes of an EP memory bank (locations 7FDH-7FFH and OFFDH-OFFFH). User program should not be single-stepped or auto-stepped through these locations.
7. Since I/O configuration is determined by external hardware rather than software, I/O modes may not be altered while a program is executing. (See Section VII for further details.)
8. The "ANL BUS,#nn" and "ORL BUS,#nn" instructions may not be used in the user program, as external hardware cannot properly restore these functions.
9. The memory bank select flag is not affected by the user program break sequence. Upon resuming execution with the [GO] command this flag will remain in the same state as before the preceding break. The flag may be cleared only by executing the [GO/RESET] or [SYS RST] commands.

VII. HARDWARE CONFIGURATIONS

A number of control and status lines are available to the user. All are low-power Schottky TTL-compatible signals.

- TP1 — Unused MP input.
- TP2 — Unused MP output.
- TP3 — User program suspended. Low when EP running user code. High when halted or running mini-monitors.
- TP4 — Breakpoint encountered. Normally low. High-level pulse generated when breakpoint passed. Useful for triggering logic analyzers, oscilloscopes, etc.
- TP5 & TP6 — Memory matrix mode control. Select program vs. data RAM, link mapping configuration, etc. (See Appendix B for details.)
- TP7 — Bus control. Low when MP controls common memory buses. High when EP controls memory buses.

The HSE-49 emulator hardware is designed to allow the user to reconfigure the system for a wide variety of different applications by installing or removing jumper wires or additional components. The schematics in Appendix A show the components needed for a variety of different configurations. In general, not all of the devices are required (or allowed) for any one configuration. The devices which are required are included in the following description.

The types of options allowed are divided below into several general classes and subdivided into mutually-independent features. Within some of these features there are numbered, mutually exclusive configurations; i.e., the serial interface (if desired) may use either

current-loop or RS-232C current buffers, but not both at one time.

Standard Operating Configuration

(Minimum system configurations — up to 4K program RAM; no data RAM; no serial interfaces; no execution processor I/O reconstruction.)

A. Basic 2K monitor from Appendix B:

- Install resistors R4-R6
- Install transistor Q1
- Install crystals Y1-Y2
- Install capacitors C5-C38
- Install switches S1-S33
- Install displays DS1-DS8
- Install IC1-IC2
- Install RP3-RP5
- Install IC6-IC7
- Install RP8
- Install IC9
- Install IC15-IC20
- Install IC25-IC30
- Install IC34
- Install IC36-IC38
- Install A1-A2
- Install B1-B2
- Install C1-C3
- Install jumpers 13-15
- Install jumpers 17-18
- Install jumper 20

B. Expansion 2K monitor:

- Install IC14
- Remove jumper 17

Serial Interface Buffer Selection

A. Current loop serial interfaces (4N46s) installed for use with full Inteltec® Model 800 development system TTY port.

- Install IC21-IC22
- Install resistor R1-R3
- Install jumpers 4-9
- (Remove RS-232 jumpers)

B. RS-232C serial interfaces (MC1488 and MC1489) installed for use with CRT as output device for data dumps:

- Install IC23-IC24
- Install jumpers 1-3
- Install jumpers 10-11
- (Remove current-loop jumpers)

External Data RAM Address Decoding Scheme for Execution Processor

A. Up to 16 pages of on-board external data RAM installed for execution processor (addresses 0 through

0FFFH = 4K bytes); port 2 used for addressing pages 0 through 15:

- Install jumpers 21-25
- Install jumper 27
- Install A5-A8
- Install B5-B8
- Install C5-C8

B. One page of on-board external data RAM installed for execution processor (addresses 0 through 0FFFH); port 2 not used for data addressing:

- Install jumper 26
- Install jumper 28
- Install A5
- Install B5
- Install C5

Connect the outputs of IC20, pins 7, 9, 10, & 11 to the inputs of a 74LS21 AND gate (not shown). Connect the output to CE and CS inputs of A5-C5. (Note: these signals are all present at jumpers 21-24 on the schematics.)

Reconstructing I/O for Execution Processor

A. Application of port 2, pins P23-P20:

- (1) Using P23-P20 for latched output data (used with "OUTL P2,A", "ANL P2,#data", and "ORL P2,#data" instructions):

Install IC31

- (2) Using P23-P20 for interfacing to an 8243 in user's prototype:

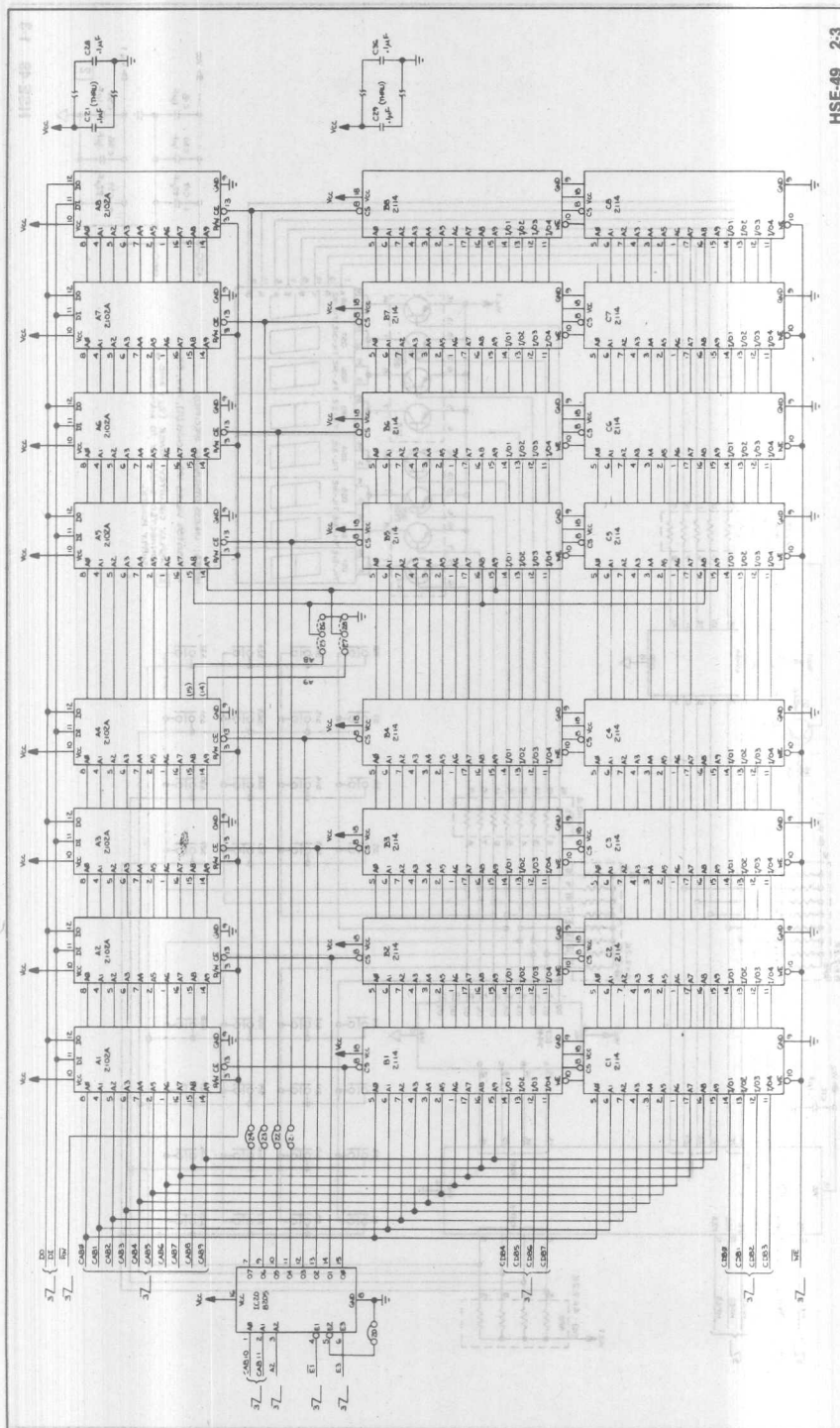
Connect D3-D0 pins on IC31 socket to corresponding Q3-Q0 pins.

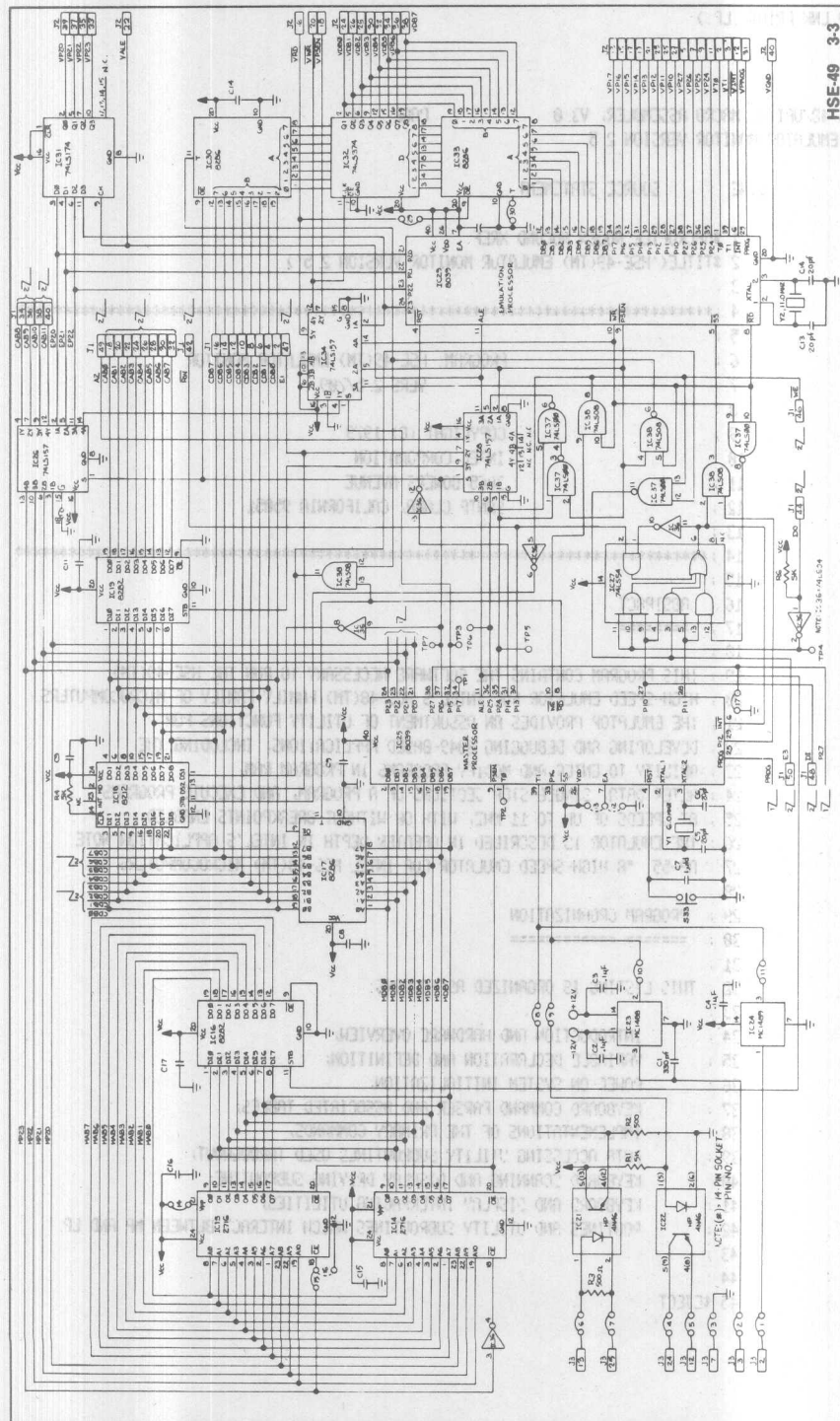
B. Application of execution processor BUS:

- (1) Use of BUS as latched output port ("OUTL BUS,A");

Install IC32







ASM48 HSE49 LNK PRINT(.LP.)

ISIS-II MCS-48/UP1-41 MACRO ASSEMBLER, V3.0
 HSE-49(TM) EMULATOR MONITOR VERSION 2.5

PAGE 1

LOC	OBJ	LINE	SOURCE STATEMENT
		1	\$MACROFILE NOGEN NOCOND XREF
		2	\$TITLE('HSE-49(TM) EMULATOR MONITOR VERSION 2.5')
		3	.
		4	*****
		5	.
		6	PROGRAM: HSE-49(TM) EMULATOR MONITOR
		7	VERS 2.5/789
		8	.
		9	COPYRIGHT (C) 1979
		10	INTEL CORPORATION
		11	3065 BOWERS AVENUE
		12	SANTA CLARA, CALIFORNIA 95051
		13	.
		14	*****
		15	.
		16	ABSTRACT
		17	*****
		18	.
		19	THIS PROGRAM CONTAINS THE SOFTWARE NECESSARY TO RUN THE HSE-49(TM)
		20	HIGH-SPEED EMULATOR FOR INTEL'S MCS-48(TM) FAMILY FAMILY OF MICROCOMPUTERS.
		21	THE EMULATOR PROVIDES AN ASSORTMENT OF UTILITY FUNCTIONS FOR
		22	DEVELOPING AND DEBUGGING 8048-BASED APPLICATIONS, INCLUDING THE
		23	ABILITY TO ENTER AND MODIFY PROGRAMS IN PROGRAM RAM.
		24	ALTER DATA, SINGLE-STEP SECTIONS OF A PROGRAM, AND EXECUTE PROGRAMS
		25	AT SPEEDS OF UP TO 11 MHz, WITH OR WITHOUT BREAKPOINTS ENABLED.
		26	THE EMULATOR IS DESCRIBED IN GREATER DEPTH IN INTEL'S APPLICATION NOTE
		27	AP-55 "A HIGH-SPEED EMULATOR FOR INTEL MCS-48(TM) MICROCOMPUTERS."
		28	.
		29	PROGRAM ORGANIZATION
		30	*****
		31	.
		32	THIS LISTING IS ORGANIZED AS FOLLOWS:
		33	.
		34	INTRODUCTION AND HARDWARE OVERVIEW;
		35	VARIABLE DECLARATION AND DEFINITION;
		36	POWER-ON SYSTEM INITIALIZATION;
		37	KEYBOARD COMMAND PARSER AND ASSOCIATED TABLES;
		38	IMPLEMENTATIONS OF THE PRIMARY COMMANDS;
		39	DATA ACCESSING UTILITY SUBROUTINES USED THROUGHOUT;
		40	KEYBOARD SCANNING AND DISPLAY DRIVING SUBROUTINES;
		41	KEYBOARD AND DISPLAY INTERFACING UTILITIES;
		42	ROUTINES AND UTILITY SUBROUTINES WHICH INTERACT BETWEEN MP AND EP.
		43	.
		44	.
		45	REJECT

LOC	OBJ	LINE	SOURCE STATEMENT	THIRTYTWO 320002	LOC	OBJ
46			;			
47			INTRODUCTION AND HARDWARE OVERVIEW			
48			=====			
49			;			
50			THE EMULATOR DESIGN USES TWO MICROPROCESSORS. ONE PROCESSOR CONTROLS			
51			SYSTEM STATUS, INTERPRETS MONITOR COMMANDS, AND COMMUNICATES			
52			WITH THE OUTSIDE WORLD THROUGH THE ON-BOARD KEYBOARD, DISPLAY, SERIAL			
53			INTERFACES, CONTROL SIGNALS, ETC.			
54			A SECOND PROCESSOR IS USED TO ACTUALLY			
55			EXECUTE THE USER'S PROGRAM UNDER THE CONTROL OF THE FIRST.			
56			THESE PROCESSORS ARE REFERRED TO			
57			THROUGHOUT THIS PROGRAM AS THE MASTER PROCESSOR (MP) AND EXECUTION			
58			PROCESSOR (EP) RESPECTIVELY.			
59			;			
60			THE PROGRAM IN THIS LISTING IS EXECUTED BY THE MASTER PROCESSOR.			
61			AT THE END OF THIS LISTING ARE SEVERAL SHORT "MINI-MONITOR OVERLAYS"			
62			WHICH THE EXECUTION PROCESSOR EXECUTES WHEN INTERACTION BETWEEN THE			
63			TWO PROCESSORS IS NECESSARY			
64			;			
65			THIS PROGRAM WAS WRITTEN USING A NUMBER OF MACROS TO HANDLE THE ALLOCATION			
66			OF MPU RESOURCES (WORKING REGISTERS, INTERNAL RAM, AND MP MONITOR ROM			
67			FOR CODE AND DATA STORAGE). THESE MACRO DEFINITIONS ARE INCLUDED IN A FILE			
68			NAMED "ALLOC.MAC." AND ARE PRINTED IN THIS LISTING FOR REFERENCE.			
69			ANOTHER SET OF MACROS IS USED TO SIMPLIFY THE ACCESSING OF VARIABLES			
70			STORED IN INTERNAL RAM (AS OPPOSED TO WORKING REGISTERS) BY USING R1 TO			
71			INDIRECTLY ADDRESS THE APPROPRIATE RAM LOCATION WHEN NECESSARY.			
72			THESE MACROS ARE INCLUDED IN "MOPCOD.MAC", AND ARE ALSO PRINTED HERE.			
73			COMPLETE UNDERSTANDING OF THESE MACROS IS NOT REQUIRED TO UNDERSTAND THE			
74			MONITOR PROGRAM. ALL LINES WHICH ACTUALLY PRODUCE OBJECT CODE APPEAR IN			
75			THE LISTING ITSELF, INDENTED TWO SPACES FROM THE NORMAL TABULATION COLUMNS.			
76			THE ACTUAL MONITOR PROGRAM FOR THE EMULATOR BEGINS AT APPROXIMATELY			
77			SOURCE LINE NUMBER 500.			
78			;			
79			LINES GENERATED BY MACRO EXPANSION ARE FLAGGED BY A PLUS SIGN ("+")			
80			IMMEDIATELY FOLLOWING THE SOURCE LINE NUMBER.			
81			A NUMBER OF LINES FROM THE VARIOUS MACRO DEFINITIONS WHICH DO NOT			
82			PRODUCE ANY OBJECT CODE ARE PROCESSED BY THE ASSEMBLER			
83			AS THESE MACROS ARE EXPANDED. WHEN THIS IS THE CASE, THESE LINES ARE			
84			SUPPRESSED FROM THE LIST FILE. AS A RESULT, THE LINE NUMBERS ARE			
85			NOT ALWAYS CONSECUTIVE WHERE A MACRO IS BEING INVOKED.			
86			;			
87			NOTE:			
88			====			
89			"SOURCE-LINE" REFERS TO THE DECIMAL NUMBERS LEFT OF EACH INSTRUCTION.			
90			AT THE END OF THE LISTING IS AN ASSEMBLY CROSS-REFERENCE TABLE INDICATING			
91			THE SEQUENTIAL SOURCE-LINE NUMBER OF ALL INSTANCES WHERE ANY VARIABLE			
92			IS DEFINED OR REFERENCED. THIS WILL BE OF GREAT ASSISTANCE IN			
93			LOCATING SPECIFIC SUBROUTINES, ETC. IN THE LISTING.			
94			;			
95			MMEMONICS COPYRIGHT (C) 1976 INTEL CORPORATION			
96			;			
97			\$EJECT			

LOC	OBJ	LINE	SOURCE STATEMENT	OBJECT STATEMENT	LOC
		98 \$	INCLUDE(=F0,ALLOC,MAC)		
0000		= 99 ?R1	SET 0		
		= 100 :			
0000		= 101 ?R0	EQU 0		
0001		= 102 ?R01	EQU 1		
0002		= 103 ?R0M	EQU 2		
0003		= 104 ?CONST	EQU 3		
0004		= 105 ?R	EQU 4		
		= 106 :			
		= 107 :	THE FOLLOWING INITIALIZES THE LINKED LIST POINTERS FOR		
		= 108 :	THE REGISTER ALLOCATION AND DEALLOCATION ROUTINES.		
		= 109 :			
0003		= 110 ?B0R2	SET 3		
0004		= 111 ?B0R3	SET 4		
0005		= 112 ?B0R4	SET 5		
0006		= 113 ?B0R5	SET 6		
0007		= 114 ?B0R6	SET 7		
0008		= 115 ?B0R7	SET 8		
		= 116 :			
0002		= 117 ?B0PNT	SET 2		
		= 118 :			
0003		= 119 ?B1R2	SET 3		
0004		= 120 ?B1R3	SET 4		
0005		= 121 ?B1R4	SET 5		
0006		= 122 ?B1R5	SET 6		
0007		= 123 ?B1R6	SET 7		
0008		= 124 ?B1R7	SET 8		
		= 125 :			
0002		= 126 ?B1PNT	SET 2		
		= 127 :			
0000		= 128 ORGPG0	SET 000H		
0100		= 129 ORGPG1	SET 100H		
0200		= 130 ORGPG2	SET 200H		
0300		= 131 ORGPG3	SET 300H		
0400		= 132 ORGPG4	SET 400H		
0500		= 133 ORGPG5	SET 500H		
0600		= 134 ORGPG6	SET 600H		
0700		= 135 ORGPG7	SET 700H		
		= 136 :			
		= 137 \$EJECT			

```

= 139 ; *****
= 139 ;
= 140 ; START OF ALLOCATION MACROS
= 141 ;
= 142 ; *****
= 143 ;
= 144 ?MSAVE MACRO SYMBOL, BANK, PNTVAL
= 145 IF PNTVAL EQ 0
= 146 ERROR 2
= 147 EXITM
= 148 ENDIF
= 149 $ SAVE GEN
= 150 SYMBOL SET R&PNTVAL
= 151 $ RESTORE
= 152 ?B&BANK&PNT SET ?B&BANK&R&PNTVAL
= 153 ENDM
= 154 ;
= 155 ;
0020 = 156 ?MINDX SET 20H
= 157 ;
= 158 ?MSAVE MACRO SYMBOL, LENGTH, ADDR
= 159 $ SAVE GEN
= 160 SYMBOL EQU ADDR
= 161 $ RESTORE
= 162 ?MINDX SET ?MINDX+LENGTH
= 163 ENDM
= 164 ;
= 165 MLOCK MACRO SYMBOL, LENGTH
= 166 ?MSYMBOL EQU 3
= 167 ?MSAVE SYMBOL, LENGTH, ?MINDX
= 168 ENDM
= 169 ;
= 170 DECLARE MACRO SYMBOL, TYPE
= 171 ?MSYMBOL SET ?&TYPE
= 172 IF ?&TYPE EQ 2
= 173 ?MSAVE SYMBOL, 1, ?MINDX
= 174 EXITM
= 175 ENDIF
= 176 IF ?&TYPE EQ 0
= 177 ?MSAVE SYMBOL, 0, ?B&PNT
= 178 EXITM
= 179 ENDIF
= 180 IF ?&TYPE EQ 1
= 181 ?MSAVE SYMBOL, 1, ?B&PNT
= 182 EXITM
= 183 ENDIF
= 184 ENDM
= 185 ;
= 186 $ EJECT

```

*** INSUFFICIENT SPACE FOR CODE ON BANK PAGE ***

LOC	OBJ	LINE	SOURCE STATEMENT	INSTRUMENT 339002	LOC	OBJ
		= 187 ;	*****			
		= 188 ; REORG	MACRO TO RESET THE INSTRUCTION LOCATION COUNTER			
		= 189 ;	TO THE FIRST FREE LOCATION ON THE FIRST PAGE MODULE WILL			
		= 190 ;	FIT WITHIN			
		= 191 REORG	MACRO LOCATION			
		= 192 \$SAVE GEN				
		= 193	ORG LOCATION			
		= 194 \$RESTORE				
		= 195	ENDM			
		= 196 ;				
		= 197 ; CODEBLK	MACRO TO FIND A PAGE OF ROM			
		= 198 ;	WHICH THIS BLOCK OF CODE WILL FIT WITHIN			
		= 199 CODEBLK	MACRO LENGTH			
		= 200 \$LENGTH SET	LENGTH			
		= 201 IF	HIGH(ORGP0+LENGTH-1) EQ 0			
		= 202	REORG \$ORGP0			
		= 203 \$START SET	\$			
		= 204 EXITM				
		= 205 ENDIF				
		= 206 IF	HIGH(ORGP1+LENGTH-1) EQ 1			
		= 207	REORG \$ORGP1			
		= 208 \$START SET	\$			
		= 209 EXITM				
		= 210 ENDIF				
		= 211 IF	HIGH(ORGP2+LENGTH-1) EQ 2			
		= 212	REORG \$ORGP2			
		= 213 \$START SET	\$			
		= 214 EXITM				
		= 215 ENDIF				
		= 216 IF	HIGH(ORGP4+LENGTH-1) EQ 4			
		= 217	REORG \$ORGP4			
		= 218 \$START SET	\$			
		= 219 EXITM				
		= 220 ENDIF				
		= 221 IF	HIGH(ORGP5+LENGTH-1) EQ 5			
		= 222	REORG \$ORGP5			
		= 223 \$START SET	\$			
		= 224 EXITM				
		= 225 ENDIF				
		= 226 IF	HIGH(ORGP6+LENGTH-1) EQ 6			
		= 227	REORG \$ORGP6			
		= 228 \$START SET	\$			
		= 229 EXITM				
		= 230 ENDIF				
		= 231 IF	HIGH(ORGP7+LENGTH-1) EQ 7			
		= 232	REORG \$ORGP7			
		= 233 \$START SET	\$			
		= 234 EXITM				
		= 235 ENDIF				
		= 236 IF	HIGH(ORGP3+LENGTH-1) EQ 3			
		= 237	REORG \$ORGP3			
		= 238 \$START SET	\$			
		= 239 EXITM				
		= 240 ENDIF				
		= 241	ERROR 0 ;*** INSUFFICIENT SPACE FOR CODE ON ANY PAGE ***			

LOC	OBJ	LINE	SOURCE STATEMENT	THGHTAT2 33R02	117	107 101
		= 242	ENDM			
		= 243	;DATABLK INSERTS ONTO PAGE 3			
		= 244	DATABLK MACRO LENGTH			
-		= 245	?LENGTH SET LENGTH			
-		= 246	IF HIGH(ORGP63+LENGTH-1) EQ 3			
-		= 247	REORG %ORGP63			
-		= 248	?START SET \$			
-		= 249	EXITM			
-		= 250	ENDIF			
-		= 251	ERROR 0 ;*** INSUFFICIENT SPACE FOR DATA BLOCK ON PAGE 3 ***			
-		= 252	ENDM			
-		= 253	?SIZE PRINTS A LINE TO THE SOURCE FILE GIVING BLOCK SIZE			
-		= 254	; AND UPDATES APPROPRIATE ORGP#			
-		= 255	?SIZE MACRO BLK.PGE			
-		= 256	\$SAVE GEN			
-		= 257	SIZE SET BLK			
-		= 258	;			
-		= 259	*****			
-		= 260	IF ?LENGTH LT SIZE			
-		= 261	ERROR 0 ;*** SIZE EXCEEDS SPACE CHECKED FOR BY CODEBLK MACRO			
-		= 262	ENDIF			
-		= 263	IF HIGH(\$-1) NE HIGH(?START)			
-		= 264	ERROR 0 ;*** CODE OR DATA BLOCK ROLLED OVER PAGE BOUNDARY ***			
-		= 265	ENDIF			
-		= 266	\$RESTORE			
-		= 267	ORGP63+GE SET \$			
-		= 268	ENDM			
-		= 269	;SIZECHK CHECKS SIZE OF PRECEDING BLOCK, PRINTS SIZE TO .LSI FILE			
-		= 270	SIZECHK MACRO			
-		= 271	?SIZE %(\$-?START),%HIGH(?START)			
-		= 272	ENDM			
-		= 273	;			
-		= 274	;			
-		= 275	;RSOURCE CODE SPACE ALLOCATION SUMMARY STATEMENT			
-		= 276	RSOURCE MACRO			
-		= 277	\$SAVE LIST GEN			
-		= 278	PGSIZE SET ORGP60-000H ;BYTES USED ON PAGE 0			
-		= 279	PGSIZE SET ORGP61-100H ;BYTES USED ON PAGE 1			
-		= 280	PGSIZE SET ORGP62-200H ;BYTES USED ON PAGE 2			
-		= 281	PGSIZE SET ORGP63-300H ;BYTES USED ON PAGE 3			
-		= 282	PGSIZE SET ORGP64-400H ;BYTES USED ON PAGE 4			
-		= 283	PGSIZE SET ORGP65-500H ;BYTES USED ON PAGE 5			
-		= 284	PGSIZE SET ORGP66-600H ;BYTES USED ON PAGE 6			
-		= 285	PGSIZE SET ORGP67-700H ;BYTES USED ON PAGE 7			
-		= 286	\$EJECT			
-		= 287	\$RESTORE			
-		= 288	ENDM			
-		= 289	\$EJECT			

LOC	OBJ	LINE	SOURCE STATEMENT	TARGET STATEMENT	LOC	OBJ
		290 ;				
		291 \$	INCLUDE(:F0:MOPCOD.MAC)			
		= 292 ;				
		= 293 ;	?FORM1 MACRO FOR GENERALIZING OPCODE INSTRUCTION			
		= 294 ;				
		= 295 ?FORM1 MACRO	OPCODE, SRC			
		= 296 IF	?&SRC EQ 2			
		= 297 \$	SAVE GEN			
		= 298	MOV R1, #SRC			
		= 299	OPCODE			
		= 300 \$	RESTORE			
		= 301	EXITM			
		= 302 ENDIF				
		= 303 IF	?&SRC EQ 0 OR ?&SRC EQ 1			
		= 304 \$	SAVE GEN			
		= 305	OPCODE A, SRC			
		= 306 \$	RESTORE			
		= 307	EXITM			
		= 308 ENDIF				
		= 309 IF	?&SRC EQ 3			
		= 310 \$	SAVE GEN			
		= 311	OPCODE A, #SRC			
		= 312 \$	RESTORE			
		= 313	EXITM			
		= 314 ENDIF				
		= 315	ERROR 1			
		= 316 ENDM				
		= 317 ;				
		= 318 ;	?FORM2 MACRO FOR GENERALIZING MOVES FROM THE ACC TO A VARIABLE			
		= 319 ?FORM2 MACRO	DEST			
		= 320 IF	?&DEST EQ 2			
		= 321 \$	SAVE GEN			
		= 322	MOV R1, #DEST			
		= 323	MOV @R1, A			
		= 324 \$	RESTORE			
		= 325	EXITM			
		= 326 ENDIF				
		= 327 IF	?&DEST EQ 0 OR ?&DEST EQ 1			
		= 328 \$	SAVE GEN			
		= 329	MOV A, #DEST, R1, #DEST			
		= 330 \$	RESTORE			
		= 331	EXITM			
		= 332 ENDIF				
		= 333	ERROR 1			
		= 334 ENDM				
		= 335 ;				
		= 336 ;	?FORM3 MACRO FOR GENERALIZING MOVES FROM THE ACC TO A VARIABLE			
		= 337 ;	WHEN IT IS KNOWN THAT R1 (IF NEEDED FOR INDIRECT ADDRESSING)			
		= 338 ;	IS ALREADY PRESET.			
		= 339 ?FORM3 MACRO	DEST			
		= 340 IF	?&DEST EQ 2			
		= 341 \$	SAVE GEN			
		= 342	MOV @R1, A			
		= 343 \$	RESTORE			
		= 344	EXITM			

LOC	OBJ	LINE	SOURCE STATEMENT	SYMBOL TABLE	DISPATCH	LOC	OBJ
..		= 345	ENDIF	002.T230 000000	0000		
..		= 346	IF ?&DEST EQ 0 OR ?&DEST EQ 1	EXITM	0000		
..		= 347	\$ SAVE GEN		0000		
..		= 348	MOV DEST, A	002.T230 000000	0000		
..		= 349	\$ RESTORE	002.T230 000000	0000		
..		= 350	EXITM	EXITM	0000		
..		= 351	ENDIF		0000		
..		= 352	ERROR 1	002.T230 000000	0000		
..		= 353	ENDM	002.T230 000000	0000		
..		= 354	;	EXITM	0000		
..		= 355	?FORM4 MACRO FOR GENERALIZING 'MOV A, SRC' INSTRUCTION	002.T230 000000	0000		
..		= 356	?FORM4 MACRO SRC	002.T230 000000	0000		
..		= 357	IF ?&SRC EQ 2	002.T230 000000	0000		
..		= 358	\$ SAVE GEN		0000		
..		= 359	MOV A, #SRC	002.T230 000000	0000		
..		= 360	MOV A, #R1	002.T230 000000	0000		
..		= 361	\$ RESTORE	002.T230 000000	0000		
..		= 362	EXITM	EXITM	0000		
..		= 363	ENDIF		0000		
..		= 364	IF ?&SRC EQ 0 OR ?&SRC EQ 1	002.T230 000000	0000		
..		= 365	\$ SAVE GEN		0000		
..		= 366	MOV A, SRC	002.T230 000000	0000		
..		= 367	\$ RESTORE	002.T230 000000	0000		
..		= 368	EXITM	EXITM	0000		
..		= 369	ENDIF		0000		
..		= 370	IF ?&SRC EQ 3	002.T230 000000	0000		
..		= 371	\$ SAVE GEN	002.T230 000000	0000		
..		= 372	MOV A, #SRC	002.T230 000000	0000		
..		= 373	\$ RESTORE	002.T230 000000	0000		
..		= 374	EXITM	EXITM	0000		
..		= 375	ENDIF		0000		
..		= 376	ERROR 1	002.T230 000000	0000		
..		= 377	ENDM	002.T230 000000	0000		
..		= 378	;		0000		
..		= 379	?FORM5 MACRO FOR GENERALIZING MOVING A CONSTANT INTO A VARIABLE	002.T230 000000	0000		
..		= 380	?FORM5 MACRO DEST, CONST	002.T230 000000	0000		
..		= 381	IF ?&DEST EQ 0 OR ?&DEST EQ 1 OR ?&DEST EQ 4	002.T230 000000	0000		
..		= 382	\$ SAVE GEN		0000		
..		= 383	MOV DEST, #CONST	002.T230 000000	0000		
..		= 384	\$ RESTORE	002.T230 000000	0000		
..		= 385	EXITM	EXITM	0000		
..		= 386	ENDIF		0000		
..		= 387	IF ?&DEST EQ 2	002.T230 000000	0000		
..		= 388	\$ SAVE GEN		0000		
..		= 389	MOV R1, #DEST	002.T230 000000	0000		
..		= 390	MOV #R1, #CONST	002.T230 000000	0000		
..		= 391	\$ RESTORE	002.T230 000000	0000		
..		= 392	EXITM	EXITM	0000		
..		= 393	ENDIF		0000		
..		= 394	ERROR 1	002.T230 000000	0000		
..		= 395	ENDM	002.T230 000000	0000		
..		= 396	;	002.T230 000000	0000		
..		= 397	MMOV MACRO GENERALIZED MOVE FROM SRC TO DEST	002.T230 000000	0000		
..		= 398	MMOV MACRO DEST, SRC	002.T230 000000	0000		
..		= 399	IF ?&SRC EQ 3	002.T230 000000	0000		

LOC	OBJ	LINE	SOURCE STATEMENT	TARGET STATEMENT	LOC
		= 400	?FORM5 DEST, SRC		
		= 401	EXITM		
		= 402	ENDIF		
		= 403	IF ?&DEST EQ 4		
		= 404	?FORM1 MOV, SRC		
		= 405	EXITM		
		= 406	ENDIF		
		= 407	IF ?&SRC EQ 4		
		= 408	?FORM2 DEST		
		= 409	EXITM		
		= 410	ENDIF		
		= 411	?FORM1 MOV, SRC		
		= 412	?FORM2 DEST		
		= 413	ENDM		
		= 414	?BINOP MACRO GENERALIZES ARITHMETIC AND LOGICAL OPERATIONS		
		= 415	?BINOP MACRO OPCODE, DEST, SRC		
		= 416	IF ?&DEST EQ 4		
		= 417	?FORM1 OPCODE, SRC		
		= 418	EXITM		
		= 419	ENDIF		
		= 420	IF ?&SRC EQ 4		
		= 421	?FORM1 OPCODE, DEST		
		= 422	?FORM3 DEST		
		= 423	EXITM		
		= 424	ENDIF		
		= 425	?FORM1 MOV, SRC		
		= 426	?FORM1 OPCODE, DEST		
		= 427	?FORM3 DEST		
		= 428	ENDM		
		= 429	:MADD MACRO FOR GENERALIZING ADD INSTRUCTION		
		= 430	MADD MACRO DEST, SRC		
		= 431	?BINOP ADD, DEST, SRC		
		= 432	ENDM		
		= 433	:		
		= 434	:MADC MACRO FOR GENERALIZING ADC INSTRUCTION		
		= 435	MADC MACRO DEST, SRC		
		= 436	?BINOP ADC, DEST, SRC		
		= 437	ENDM		
		= 438	:		
		= 439	:MANL MACRO FOR GENERALIZING ANL INSTRUCTION		
		= 440	MANL MACRO DEST, SRC		
		= 441	?BINOP ANL, DEST, SRC		
		= 442	ENDM		
		= 443	:		
		= 444	:MORL MACRO FOR GENERALIZING ORL INSTRUCTION		
		= 445	MORL MACRO DEST, SRC		
		= 446	?BINOP ORL, DEST, SRC		
		= 447	ENDM		
		= 448	:		
		= 449	:MXRL MACRO FOR GENERALIZING XRL INSTRUCTION		
		= 450	MXRL MACRO DEST, SRC		
		= 451	?BINOP XRL, DEST, SRC		
		= 452	ENDM		
		= 453	:		
		= 454	:MXCH MACRO FOR GENERALIZING XCH INSTRUCTION		

LOC	OBJ	LINE	SOURCE STATEMENT	OBJECT STATEMENT	LOC	OBJ
		= 455	MXCH MACRO DEST, SRC			
		= 456	?BINOP XCH, DEST, SRC			
		= 457	ENDM			
		= 458 ;				
		= 459	?UNARY MACRO OPCODE, DEST			
		= 460	?FORM1 MOV, DEST			
		= 461	\$SAVE GEN			
		= 462	OPCODE A			
		= 463	\$RESTORE			
		= 464	?FORM3 DEST			
		= 465	ENDM			
		= 466 ;				
		= 467	MINC MACRO DEST			
		= 468	?UNARY INC, DEST			
		= 469	ENDM			
		= 470	MDEC MACRO DEST			
		= 471	?UNARY DEC, DEST			
		= 472	ENDM			
		= 473	MJNZ MACRO DEST, ADDR			
		= 474	?UNARY DEC, DEST			
		= 475	\$SAVE GEN			
		= 476	JNZ ADDR			
		= 477	\$RESTORE			
		= 478	ENDM			
		= 479 ;				
		= 480	MRL MACRO DEST			
		= 481	?UNARY RL, DEST			
		= 482	ENDM			
		= 483	MRR MACRO DEST			
		= 484	?UNARY RR, DEST			
		= 485	ENDM			
		= 486	MRC MACRO DEST			
		= 487	?UNARY RC, DEST			
		= 488	ENDM			
		= 489	MRC MACRO DEST			
		= 490	?UNARY RC, DEST			
		= 491	ENDM			
		= 492	MRL MACRO DEST			
		= 493	?UNARY RL, DEST			
		= 494	ENDM			
		= 495	MRR MACRO DEST			
		= 496	?UNARY RR, DEST			
		= 497	ENDM			
		= 498	\$EJECT			


```

499 ;
500 ;=====
501 ;=====
502 ;          BEGINNING OF PROGRAM PROPER
503 ;=====
504 ;=====
505 ;
506 ;
507 ;*****
508 ;
509 ;      ALLOCATION OF MP I/O PORTS:
510 ;
511 ;*****
512 ;
513 ;      BUS          ;USED FOR BIDIRECTIONAL ADDRESS AND DATA TRANSFERS
514 ;      P1          ;USED AS INDIVIDUAL CONTROL OUTPUTS AND BREAK LOGIC
515 ;      P2          ;HIGH-ORDER ADDRESS AND ADDRESS SPACE SELECTION
516 ;
000E 517 PDIGIT EQU      P7      ;USED TO ENABLE CHARACTERS AND STROBE ROWS OF KEYBOARD
000D 518 PSEGH1 EQU     P6      ;USED TO TURN ON HI SEGMENTS OF CURRENTLY ENABLED DIGIT
000C 519 PSEGLO EQU     P5      ;PORT FOR LOWER FOUR SEGMENTS
000B 520 PINPUT EQU     P4      ;PORT USED TO SCAN FOR KEY CLOSURES
521 ;
522 ;*****
523 ;
524 ;      INDIVIDUAL PINS OF PORT 1 USED AS FOLLOWS:
525 ;
526 ;*****
527 ;
0001 528 ENDRAM EQU      0000001D ;P10 - HI ENABLES BREAK ON BREAK RAM OUTPUT SIGNAL
0002 529 ENBLNK EQU      00000010B ;P11 - HI ENABLES BREAK ON RD OR WR TO LINK BY EP
530 ;      (NOTE: P11 & P10 BOTH HI ENABLES
531 ;      BREAK ON ANY EP INSTRUCTION CYCLE)
0004 532 EPSSTP EQU      00000100B ;P12 - LO FORCES EP SS INPUT LOW
533 ;      HI GATES BREAKPOINT FLIP-FLOP TO EP SS INPUT.
0008 534 CLRBEFF EQU      00001000B ;P13 - LO CLEARS BREAK FLIP-FLOP
535 ;      AND ENABLES WR CONTROL TO BREAKPOINT RAM.
0010 536 EPRSET EQU      00010000B ;P14 - HI RESETS EP
0020 537 MODOUT EQU      00100000B ;P15 - LO WHEN EP IS EXECUTING USER PROGRAM,
538 ;      HI WHEN EP FROZEN OR RUNNING OVERLAYS.
0040 539 TTYOUT EQU      01000000B ;P16 - SERIAL OUTPUT TO TTY OR CRT
540 ;      ;P17 - UNUSED
541 ;
542 $EJECT

```

LOC	OBJ	LINE	SOURCE STATEMENT	INTEGRATE 300002	LD 103
		543	*****		
		544	*****		
		545	INDIVIDUAL PINS OF PORT 2 USED AS FOLLOWS		
		546	*****		
		547	*****		
		548	*****		
		549	P23-P20 :ADR11-ADR8 FOR ACCESSING PROGRAM OR DATA RAM ARRAY		
		550	*****		
0010		551 M0	EQU 00010000B :P24 - MEMORY MATRIX CONTROL PIN 0		
0020		552 M1	EQU 00100000B :P25 - MEMORY MATRIX CONTROL PIN 1		
0040		553 MPUSEL	EQU 01000000B :P26 - HIGH WHEN MP IN CONTROL OF COMMON MEM ARRAY,		
		554	LOW WHEN EP IN CONTROL		
0080		555 EXPMON	EQU 10000000B :P27 - JUMPED TO GROUND FOR STANDARD MONITOR,		
		556	FLOATING WHEN EXPANSION MONITOR PRESENT		
		557	*****		
		558	*****		
		559	WHEN MP IN CONTROL OF MEMORY MATRIX M1-M0 USED AS FOLLOWS		
		560	*****		
		561	M1 M0 MODE		
		562	0 0 PROGRAM RAM ARRAY ENABLED FOR READ & WRITE		
		563	0 1 DATA RAM ARRAY ENABLED FOR READ & WRITE		
		564	1 X LINK REGISTER ENABLED FOR READ, RAM ARRAYS DISABLED		
		565	(NOTE: LINK REGISTER ALWAYS ENABLED FOR MP WRITES)		
		566	*****		
		567	WHEN EP IN CONTROL OF MATRIX M1-M0 USED AS FOLLOWS		
		568	*****		
		569	M1 M0 MODE		
		570	0 X EP PSEN FETCHES FROM LINK REGISTER (USED TO FORCE OPCODES)		
		571	1 0 EP PSEN FETCHES FROM PROGRAM RAM ARRAY		
		572	EP RD & WR CONTROL DATA RAM ARRAY		
		573	1 1 EP PSEN FETCHES FROM PROGRAM RAM ARRAY		
		574	RD & WR CONTROL LINK REGISTER		
		575	*****		
		576	\$EJECT		

LOC	OBJ	LINE	SOURCE STATEMENT	THIRTYTWO THREE	LOC	OBJ
		577	*****			
		578	*****			
		579	*****			
		580	SYSTEM CONSTANT DEFINITIONS			
		581	*****			
		582	*****			
		583	*****			
0008		584	DECLARE CHARNO.CONST ,NUMBER OF DIGITS IN DISPLAY AND ROWS OF KEYS			
		598	CHARNO EQU 8			
		599	*****			
0004		600	DECLARE NCOLS.CONST ,LESSER DIMENSION OF KEYBOARD MATRIX			
		614	NCOLS EQU 4			
		615	*****			
0008		616	DECLARE DEDNCE.CONST ,NUMBER OF SUCESSIVE SCANS BEFORE KEY CLOSURE ACCEPTED			
		630	DEDNCE EQU 8			
		631	*****			
0017		632	DECLARE OVSZ.CONST ,SIZE OF LARGEST MINI-MONITOR OVERLAY FOR EP			
		646	OVSZ EQU 23			
		647	*****			
0010		648	DECLARE BUFLN.CONST ,LENGTH OF HEX FORMAT XMIT BUFFER (MAX RECORD LENGTH)			
		662	BUFLN EQU 16			
		663	*****			
		664	*****			
		665	*****			
		666	UTILITY CONSTANT DECLARATIONS			
		667	*****			
		668	*****			
		669	*****			
0000		670	DECLARE ZERO.CONST			
		684	ZERO EQU 0			
0001		685	DECLARE PLUS1.CONST			
		699	PLUS1 EQU 1			
0003		700	DECLARE PLUS3.CONST			
		714	PLUS3 EQU 3			
		715	DECLARE NEG1.CONST			
FFFF		729	NEG1 EQU -1			
		730	*****			
		731	\$EJECT			

```

732 ;
733 ;*****
734 ;
735 ;      BANK 0 REGISTER ALLOCATION:
736 ;
737 ;*****
738 ;
0002 739 DECLARE LDATA,RB0      ;DATA USED BY LOGICAL ADDRESSING READ/WRITE UTILITIES
754+   LDATA SET R2
0003 756 DECLARE KEY,RB0      ;HOLDS KEYCODE RETURNED FROM KBD INPUT ROUTINE
769+   KEY SET R3
0004 773 DECLARE ITMP,RB0     ;COUNTER USED AS AN INDEX IN PARSER ROUTINE
786+   ITMP SET R4
0005 790 DECLARE CHKSUM,RB0   ;CHECKSUM OF DATA BYTES TRANSMITTED IN HEX FILE FORMAT
803+   CHKSUM SET R5
0006 807 DECLARE DSPTMP,RB0   ;TEMPORARY STORAGE FOR DISPLAY PATTERNS IN 'DSPACC'
820+   DSPTMP SET R6
0007 824 DECLARE XPCODE,RB0   ;EXPANSION MONITOR ROUTINE CODE NUMBER
837+   XPCODE SET R7
841 ;
842 ;*****
843 ;
844 ;      BANK 1 REGISTER ALLOCATION
845 ;
846 ;*****
847 ;
0002 848 DECLARE ROTPAT,RB1     ;USED TO HOLD INPUT PATTERN BEING ROTATED THROUGH CV
865+   ROTPAT SET R2
0003 869 DECLARE ROTCNT,RB1   ;COUNTS NUMBER OF BITS ROTATED THROUGH CV
886+   ROTCNT SET R3
0004 890 DECLARE LASTKY,RB1   ;HOLDS KEY POSITION OF LAST KEY DEPRESSION DETECTED
907+   LASTKY SET R4
0005 911 DECLARE CURDIG,RB1   ;HOLDS POSITION OF NEXT CHARACTER TO BE DISPLAYED
928+   CURDIG SET R5
0006 932 DECLARE KEYFLG,RB1   ;FLAG TO DETECT WHEN ALL KEYS ARE RELEASED
949+   KEYFLG SET R6
953 ;      ;(REGISTER 7 NOT USED FOR PRIMARY MONITOR)
954 ;
955 ;*****
956 $EJECT

```


LOC	OBJ	LINE	SOURCE STATEMENT	STATEMENT	LOC	OBJ
		957 ;				
		958 :*****				
		959 ;				
		960 : DATA RAM ALLOCATION				
		961 ;				
		962 :*****				
		963 ;				
0020		964 DECLARE EPACC,RAM	; STORAGE IN MP FOR EP ACCUMULATOR			
		969+ EPACC EQU 32				
		973 DECLARE EPPSW,RAM	; STORAGE IN MP FOR EP PROGRAM STATUS WORD			
0021		978+ EPPSW EQU 33				
		982 DECLARE EPTIMR,RAM	; STORAGE IN MP FOR EP TIMER/COUNTER REGISTER			
0022		987+ EPTIMR EQU 34				
		991 DECLARE EPR0,RAM	; STORAGE IN MP FOR EP REGISTER 0 OF BANK 0			
0023		996+ EPR0 EQU 35				
		1000 DECLARE EPPCLO,RAM	; STORAGE IN MP FOR LOW BYT OF EP PROGRAM COUNTER			
0024		1005+ EPPCLO EQU 36				
		1009 DECLARE EPPCHI,RAM	; STORAGE IN MP FOR HIGH NIBBLE OF EP PROGRAM COUNTER			
0025		1014+ EPPCHI EQU 37				
		1018 DECLARE HBITLO,RAM	; PARAMETER 1 FOR SERIAL LINK DATA RATE GENERATOR			
0026		1023+ HBITLO EQU 38				
		1027 DECLARE HBITHI,RAM	; PARAMETER 2 FOR SERIAL LINK DATA RATE GENERATOR			
0027		1032+ HBITHI EQU 39				
		1036 DECLARE DSPTIM,RAM	; PARAMETER FOR AUTO-STEP AND AUTO-BREAK SEQUENCING RATE			
0028		1041+ DSPTIM EQU 40				
		1045 DECLARE VERSND,RAM	; MONITOR VERSION NUMBER			
0029		1050+ VERSND EQU 41				
		1054 DECLARE HREGA,RAM	; (UNUSED)			
002A		1059+ HREGA EQU 42				
		1063 DECLARE HREGD,RAM	; (UNUSED)			
002B		1068+ HREGD EQU 43				
		1072 DECLARE HREGC,RAM	; (UNUSED)			
002C		1077+ HREGC EQU 44				
		1081 DECLARE HREGD,RAM	; (UNUSED)			
002D		1086+ HREGD EQU 45				
		1090 DECLARE HREGC,RAM	; (UNUSED)			
002E		1095+ HREGC EQU 46				
		1099 DECLARE HREGF,RAM	; (UNUSED)			
002F		1104+ HREGF EQU 47				
		1108 DECLARE SMALO,RAM	; PRIMARY COMMAND STARTING MEMORY ADDRESS (LOW BYTE)			
0030		1113+ SMALO EQU 48				
		1117 DECLARE SMAHI,RAM	; PRIMARY COMMAND STARTING MEMORY ADDRESS (HIGH BYTE)			
0031		1122+ SMAHI EQU 49				
		1126 DECLARE EMALO,RAM	; PRIMARY COMMAND ENDING MEMORY ADDRESS (LOW BYTE)			
0032		1131+ EMALO EQU 50				
		1135 DECLARE EMAHI,RAM	; PRIMARY COMMAND ENDING MEMORY ADDRESS (HIGH BYTE)			
0033		1140+ EMAHI EQU 51				
		1144 DECLARE MEMLO,RAM	; THIRD PARSER PARAMETER & HEX RECORD ADDRESS (LOW)			
0034		1149+ MEMLO EQU 52				
		1153 DECLARE MEMHI,RAM	; THIRD PARSER PARAMETER & HEX RECORD ADDRESS (HIGH)			
0035		1158+ MEMHI EQU 53				
		1162 DECLARE BCODE,RAM	; PRIMARY COMMAND NUMBER FROM PARSER TABLES (0-9)			
0036		1167+ BCODE EQU 54				
		1171 DECLARE TYPE,RAM	; PRIMARY COMMAND MODIFIER/OPTION (0-5)			
0037		1176+ TYPE EQU 55				

LOC	OBJ	LINE	SOURCE STATEMENT	THINSTAT2 330402	103	001
		1180	DECLARE NUMCON, RAM ; MAX. NUMBER OF PARAMETERS ALLOWED FOR SELECTED COMMAND			
0030		1185+	NUMCON EQU 56	001 000	001	0000
		1189	DECLARE OPTION, RAM ; INDEX POINTER USED IN SEARCHING PARSER TABLES			
0039		1194+	OPTION EQU 57	001 001	001	0000
		1198	DECLARE NEXTPL, RAM ; CHARACTER POSITION FOR DISPLAY UTILITIES TO WRITE NEXT			
003A		1203+	NEXTPL EQU 58	001 002	001	0000
		1207	DECLARE KBDBUF, RAM ; POSITION OF KEY DEBOUNCED BY SCANNING SUBROUTINE			
003B		1212+	KBDBUF EQU 59	001 003	001	0000
		1216	DECLARE KEYLOC, RAM ; INCREMENTED AS SUCCESSIVE KEY LOCATIONS SCANNED			
003C		1221+	KEYLOC EQU 60	001 004	001	0000
		1225	DECLARE NREPTS, RAM ; KEEPS TRACK OF SUCCESSIVE READS OF SAME KEYSTROKE			
003D		1230+	NREPTS EQU 61	001 005	001	0000
		1234	DECLARE ASAVL, RAM ; HOLDS ACCUMULATOR VALUE DURING SERVICE ROUTINE			
003E		1239+	ASAVE EQU 62	001 006	001	0000
		1243	DECLARE RDELAY, RAM ; COUNTER DECREMENTED WHEN AUTO-STEP DELAY IN PROGRESS			
003F		1248+	RDELAY EQU 63	001 007	001	0000
		1252	DECLARE STRTMP, RAM ; INDEX POINTER FOR DISPLAY CHARACTER STRING ACCESSING			
0040		1257+	STRTMP EQU 64	001 008	001	0000
		1261	DECLARE BUFCNT, RAM ; COUNT OF DATA BYTES IN HEX FORMAT RECORD BUFFER			
0041		1266+	BUFCNT EQU 65	001 009	001	0000
		1270	DECLARE RECTYP, RAM ; TYPE OF HEX FORMAT RECORD (0 OR 1)			
0042		1275+	RECTYP EQU 66	001 010	001	0000
		1279	DECLARE B, RAM ; BIT COUNTER FOR ASCII SERIAL I/O UTILITY SUBROUTINES			
0043		1284+	B EQU 67	001 011	001	0000
		1288	DECLARE REGC, RAM ; CHARACTER BEING SHIFTED DURING SERIAL I/O PROCESS			
0044		1293+	REGC EQU 68	001 012	001	0000
		1297	DECLARE H, RAM ; COUNTER IN SOFTWARE DELAY DATA RATE GENERATOR			
0045		1302+	H EQU 69	001 013	001	0000
		1306 ;		001 014	001	0000
		1307	MBLOCK SEGMAP, CHARNO ; REGISTER ARRAY FOR DISPLAY PATTERNS			
0046		1311+	SEGMAP EQU 70	001 015	001	0000
		1314 ;		001 016	001	0000
		1315	MBLOCK OVBUFF, OVSZ ; LOW-ORDER USER PROGRAM DURING MINI-MONITOR OVERLAYS			
004E		1319+	OVBUFF EQU 78	001 017	001	0000
		1322 ;		001 018	001	0000
		1323	MBLOCK HEXBUF, BUFLN ; ALLOCATE BLOCK OF RAM FOR USE AS HEX RECORD BUFFER			
0065		1327+	HEXBUF EQU 101	001 019	001	0000
		1330 ;		001 020	001	0000
		1331	\$EJECT	001 021	001	0000

```

0300 1332 00000000 DATADLK 40
1337+ 0000 768
1341 ; INVALS TABLE OF CONSTANTS TO BE LOADED INTO MP INTERNAL RAM VARIABLES
1342 ; AS PART OF SYSTEM INITIALIZATION PROCEDURE:
1343 ; CHARACTER POSITION FOR DISPLAY UTILITIES TO
1344 ; INITIAL VALUE VARIABLE TYPE
1345 ;=====
0300 00 1346 INVALS: DB 00H ; ROTPAT RB1
0301 00 1347 DB 00H ; ROTCNT RB1
0302 00 1348 DB 00H ; LASTKY RB1
0303 00 1349 DB 00H ; CHARN0 ; CURD1G RB1
0304 00 1350 DB 00H ; KEYFLG RB1
0305 00 1351 DB 00H ; <REG7> RB1
0306 00 1352 DB 00H ; EPACC RAM
0307 01 1353 DB 01H ; LPPSW RAM
0308 00 1354 DB 00H ; EPTMR RAM
0309 00 1355 DB 00H ; EPRO RAM
030A 00 1356 DB 00H ; EPPCLO RAM
030B 00 1357 DB 00H ; EPPCHI RAM
030C 93 1358 DB 93H ; HBITLO RAM
030D 04 1359 DB 04H ; HBITHI RAM
030E 20 1360 DB 20H ; DSPTIM RAM
030F 25 1361 DB 25H ; VERSNO RAM
0310 00 1362 DB 00H ; HREGA RAM
0311 00 1363 DB 00H ; HREGB RAM
0312 00 1364 DB 00H ; HREGC RAM
0313 00 1365 DB 00H ; HREGD RAM
0314 00 1366 DB 00H ; HREGE RAM
0315 00 1367 DB 00H ; HREGF RAM
0316 00 1368 DB 00H ; SMALO RAM
0317 00 1369 DB 00H ; SMAHI RAM
0318 FF 1370 DB 0FFH ; EMALO RAM
0319 0F 1371 DB 0FH ; EMAHI RAM
031A 00 1372 DB 00H ; MEMLO RAM
031B 00 1373 DB 00H ; MEMHI RAM
031C 00 1374 DB 00H ; BCODE RAM
031D 04 1375 DB 04H ; TYPE RAM
031E 01 1376 DB 01H ; NUMCON RAM
031F 00 1377 DB 00H ; OPTION RAM
0320 00 1378 DB 00H ; CHARN0 ; NEXTPL RAM
0321 FF 1379 DB 0FFH ; KDBUF RAM
0322 00 1380 DB 00H ; KEYLOC RAM
0023 1381 NOVALS EQU $-INVALS
1382 SIZECHK
0023 1385+ SIZE SET 35
1386+
1387+ ;*****
1396 $EJECT

```

LOC	OBJ	LINE	SOURCE STATEMENT
		1397 \$	INCLUDE(:(F0:PARSER.MOD)
		=1398	CODEBLK 45
0000		=1403+	ORG 0
		=1407 ; INIT	INITIALIZES PROCESSOR REGISTERS
		=1408 ;	AND RAM LOCATIONS TO DEFINED VALUES.
0000 C5		=1409 INIT:	SEL R00
0001 BF00		=1410	MOV XPCODE, #0
0003 74D1		=1411	CALL XPTST
0005 27		=1412	CLR A
0006 3D		=1413	MOVD PSEGLO, A
0007 3E		=1414	MOVD PSEGH1, A
0008 D81A		=1415	MOV R0, #1AH ; START AT KD1 (REG2) = RAM LOC 1AH
000A B923		=1416	MOV R1, #LOW NOVALS
000C BA00		=1417	MOV R2, #LOW INVALS
000E FA		=1418 INITLP:	MOV A, R2
000F E3		=1419	MOV3 A, #A
0010 A0		=1420	MOV #R0, A
0011 18		=1421	INC R0
0012 1A		=1422	INC K2
0013 E90E		=1423	DJNZ R1, INITLP
0015 55		=1424	STR T
0016 744F		=1425	CALL EPBRK
0018 1888		=1426	MOV R0, #LOW(OVIBAS+OVSIZE)
001A 746A		=1427	CALL OVLOAD
001C 54E5		=1428	CALL COMFIL
001E D937		=1429	MOV R1, #TYPE
0020 11		=1430	INC #R1
0021 34F2		=1431	CALL INCSMA
0023 54E5		=1432	CALL COMFIL
0025 99E1		=1433	ANL PL, # (NOT EPSET) ; REMOVE EP RESET SIGNAL
0027 0429		=1434	JMP MAIN
		=1435 ;	
		=1436	SIZECHK
0029		=1439+ SIZE SET 41	
		=1440+;	
		=1441+; *****	
		=1450 \$EJECT	

LOC	OBJ	LINE	SOURCE STATEMENT	TARGET STATEMENT	LOC	OBJ
=1451 ;				INITIALIZE PROCESSOR REGISTERS		
=1452 ;			KEYBOARD LAYOUT:	CODES		
=1453 ;			=====	8		
=1454 ;				INITIALIZE PROCESSOR REGISTERS		
=1455 ;				INITIALIZE PROCESSOR REGISTERS		
=1456 ;			!!	!!		
=1457 ;			LIST !!GO/RESET!! GO !!EXAM/CHN!!	!!D !!E !!F !!		
=1458 ;			!!	!!		
=1459 ;						
=1460 ;						
=1461 ;			!!PROG BRK!!PROG MEM!!REGISTER!!	!!A !!B !!C !!D !!E !!F !!		
=1462 ;			UPLOAD !!	!!8 !!9 !!A !!B !!C !!D !!E !!F !!		
=1463 ;			!!AUTO STP!!SING STP!! NO BRK !!	!!		
=1464 ;						
=1465 ;						
=1466 ;			!!DATA BRK!!DATA MEM!!	!!		
=1467 ;			DNLOAD !!	!!4 !!5 !!6 !!7 !!8 !!9 !!A !!B !!C !!D !!E !!F !!		
=1468 ;			!!AUTO BRK!!WITH BRK!!	!!		
=1469 ;						
=1470 ;						
=1471 ;			!!	!!		
=1472 ;			FILL !!HARD REG!! NEXT/, !! END/	!!0 !!1 !!2 !!3 !!4 !!5 !!6 !!7 !!8 !!9 !!A !!B !!C !!D !!E !!F !!		
=1473 ;			!!	!!		
=1474 ;						
=1475 ;						
=1476			\$JECT			

LOC	OBJ	LINE	SOURCE STATEMENT	TRANSLATED STATEMENT	LOC	OBJ
		=1477 ;				
		=1478 ;	THE FOLLOWING EQUATES DETERMINES HOW THE PARSER INTERPRETS			
		=1479 ;	VALUES RETURNED BY THE KEYBOARD SCANNING INPUT ROUTINE			
		=1480 ;	WHEN THE VARIOUS KEYS OF THE KEYBOARD ARE PRESSED.			
		=1481 ;				
		=1482 ;				
		=1483 ; KEY0 EQU 00H	VALUE RETURNED FOR EACH KEY OF KEYBOARD MATRIX			
		=1484 ; KEY1 EQU 01H	BY KEYBOARD SCANNING SUBROUTINE "KBDIN".			
		=1485 ; KEY2 EQU 02H				
		=1486 ; KEY3 EQU 03H				
		=1487 ; KEY4 EQU 04H	! 1C ! 1D ! 1E ! 1F ! ! 0C ! 0D ! 0E ! 0F !			
		=1488 ; KEY5 EQU 05H				
		=1489 ; KEY6 EQU 06H	! 18 ! 19 ! 1A ! 1B ! ! 08 ! 09 ! 0A ! 0B !			
		=1490 ; KEY7 EQU 07H				
		=1491 ; KEY8 EQU 08H	! 14 ! 15 ! 16 ! 17 ! ! 04 ! 05 ! 06 ! 07 !			
		=1492 ; KEY9 EQU 09H				
		=1493 ; KEYA EQU 0AH	! 10 ! 11 ! 12 ! 13 ! ! 00 ! 01 ! 02 ! 03 !			
		=1494 ; KEYC EQU 0BH				
		=1495 ; KEYD EQU 0CH				
		=1496 ; KEYE EQU 0DH				
		=1497 ; KEYF EQU 0EH				
		=1498 ; KEYF EQU 0FH				
0010		=1499 KEYFIL EQU 10H	:[FILL COMMAND]			
0012		=1500 KEYNXT EQU 12H	:[NEXT/]			
0013		=1501 KEYEND EQU 13H	:[END/]			
0014		=1502 KEYREL EQU 14H	:[DOWNLOAD COMMAND]			
0015		=1503 KEYPAT EQU 15H	:[AUTOBREAK MODIFIER]			
0016		=1504 KEYDM EQU 16H	:[DATA MEMORY MODIFIER]			
0017		=1505 KEYCLR EQU 17H	:[CLEAR/PREVIOUS]			
0018		=1506 KEYREC EQU 18H	:[UPLOAD COMMAND]			
0019		=1507 KEYTRA EQU 19H	:[AUTOSTEP MODIFIER]			
001A		=1508 KEVPM EQU 1AH	:[PROGRAM MEMORY MODIFIER]			
001B		=1509 KEYREG EQU 1BH	:[REGISTER MEMORY MODIFIER]			
001C		=1510 KEVLST EQU 1CH	:[FORMATTED DATA OUTPUT COMMAND]			
001D		=1511 KGORES EQU 1DH	:[GO FROM RESET STATE COMMAND]			
001E		=1512 KEYGD EQU 1EH	:[GO COMMAND]			
001F		=1513 KEYMOD EQU 1FH	:[EXAMINE/MODIFY COMMAND]			
0008		=1514 KSETB EQU 0BH	:[SET BREAKPOINT COMMAND]			
000C		=1515 KCLRB EQU 0CH	:[CLEAR BREAKPOINT COMMAND]			
		=1516 ;				
		=1517 ;				
0019		=1518 PERK EQU 19H	:[PROGRAM BREAKPOINT MEMORY MODIFIER]			
0015		=1519 DBRK EQU 15H	:[DATA BREAKPOINT MEMORY MODIFIER]			
0011		=1520 RINT EQU 11H	:[HARDWARE REGISTER MEMORY MODIFIER]			
001B		=1521 NOBRK EQU 1BH	:[WITHOUT BREAKPOINTS MODIFIER]			
001C		=1522 WBRK EQU 1CH	:[WITH BREAKPOINTS ENABLED MODIFIER]			
001A		=1523 SING EQU 1AH	:[SINGLE STEP MODIFIER]			
		=1524 ;				
		=1525 \$EJECT				

LOC	OBJ	LINE	SOURCE STATEMENT	INSTRUMENT	DISASSEMBLY	LOC	OBJ
		=1526	CODEBLK 160				
0029		=1531+	ORG 41				
		=1535 ;	MAIN OUTPUT_MESSAGE(COMMAND_PROMPT)				
		=1536 ;	CALL INPUT_BYTE(KEY)				
		=1537 ;	MAIN2 IF THE KEY=END GO TO MAIN				
		=1538 ;					
0029 B701		=1539 MAIN:	MOV XPCODE, #1	H00	003	B701	
002B 7401		=1540	CALL XPTST	H20	003	7401	
002D 2301		=1541	MOV A, #1	H50	003	2301	
002F 3400		=1542	CALL OUTUTL	H30	003	3400	
0031 14EC		=1543	CALL INPKEY	H40	003	14EC	
0033 FB		=1544 MAIN2:	MOV A, KEY	H20	003	FB	
0034 D313		=1545	XRL A, #KEYEND	H30	003	D313	
0036 C629		=1546	JZ MAIN	H70	003	C629	
		=1547 ;					
		=1548 ;	FINDOP FIND OUT IF THE KEY PRESSED IS A LEGITIMATE COMMAND INITIATOR:				
		=1549 ;	ITMP:=CTAB	H40	003		
		=1550 ;	BCODE:=TYPE:=0	H00	003		
		=1551 ;	WHILE CTAB(ITMP)<0 /CTAB EXHAUSTED/	H00	003		
		=1552 ;	IF CTAB(ITMP)=KEY GOTO MAINA /COMMAND ENTRY FOUND IN CTAB/	H00	003		
		=1553 ;	ELSE ITMP:=ITMP+COMMAND_ENTRY_SIZE	H30	003		
		=1554 ;	BCODE:=BCODE+1	H70	003		
		=1555 ;	ENDWHILE	H01	003		
		=1556 ;	GOTO ERROR	H51	003		
0038 BC23		=1557	MOV ITMP, #CTAB	H51	003	BC23	
		=1558	MMOV BCODE, ZERO	H41	003		
0039 B936		=1569+	MOV R1, #BCODE	H01	003	B936	
003C B100		=1570+	MOV @R1, #ZERO	H01	003	B100	
		=1574	MMOV TYPE, ZERO	H51	003		
003E B937		=1585+	MOV R1, #TYPE	H01	003	B937	
0040 B100		=1586+	MOV @R1, #ZERO	H01	003	B100	
0042 FC		=1590 FINDOP:	MOV A, ITMP	H41	003	FC	
0043 E3		=1591	MOV #3, A	H01	003	E3	
0044 B2BC		=1592	JBS MERROR	H31	003	B2BC	
0046 DB		=1593	XRL A, KEY	H01	003	DB	
0047 C652		=1594	JZ MAINA	H31	003	C652	
0049 FC		=1595	MOV A, ITMP	H41	003	FC	
004A B303		=1596	ADD A, #COMSIZ	H00	003	B303	
004C AC		=1597	MOV ITMP, A	H00	003	AC	
004D B936		=1598	MOV R1, #BCODE	H01	003	B936	
004F 11		=1599	INC @R1	H11	003	11	
0050 0442		=1600	JMP FINDOP	H21	003	0442	
		=1601 ;					
		=1602 ;	OUTPUT_MESSAGE(STRCON(BCODE)) /*PROMPT FOR THE CURRENT COMMAND*/				
		=1603 ;	I:=I+1	H01	003		
		=1604 ;	OPTION:=MEM(I)	H01	003		
		=1605 ;	I:=I+1	H01	003		
		=1606 ;	NO_OF_PARAMETERS:=MEM(I)				
		=1607 ;	I:=3				
		=1608 ;					
		=1609 MAINA:	MMOV A, BCODE				
0052 B936		=1618+	MOV R1, #BCODE				
0054 F1		=1619+	MOV A, @R1				
0055 B31D		=1623	ADD A, #STRCON				
0057 3402		=1624	CALL OUTCLR				

LOC	OBJ	LINE	SOURCE STATEMENT	OBJECT STATEMENT	LOC	OBJ
0059	1C	=1625	INC ITMP	MOV A,ITMP	0059	1C
005A	FC	=1626	MOV A,ITMP	MOV A,ITMP	005A	FC
005B	E3	=1627	MOV P3 A,0A ; GET OPTION POINTER	MOV P3 A,0A	005B	E3
		=1628	MMOV OPTION,A	MMOV OPTION,A		
005C	B939	=1641+	MOV R1,#OPTION	MOV R1,#OPTION	005C	B939
005E	R1	=1642+	MOV @R1,A	MOV @R1,A	005E	R1
005F	1C	=1646	INC ITMP	INC ITMP	005F	1C
0060	FC	=1647	MOV A,ITMP	MOV A,ITMP	0060	FC
0061	E3	=1648	MOV P3 A,0A ; GET NO OF PARAMETERS	MOV P3 A,0A	0061	E3
		=1649	MMOV NUMCON,A	MMOV NUMCON,A		
0062	B938	=1662+	MOV R1,#NUMCON	MOV R1,#NUMCON	0062	B938
0064	R1	=1663+	MOV @R1,A	MOV @R1,A	0064	R1
		=1667 ;				
		=1668 ;	PARAMETER_BUFFER(0=5):=0			
		=1669 ;				
0065	B90C	=1670	MOV R1,06 ; EACH PARAM IS 2 BYTES	MOV R1,06	0065	B90C
0067	B930	=1671	MOV R0,#SMALO ; START OF PARAM BUFFERS	MOV R0,#SMALO	0067	B930
0069	B000	=1672 MAINB:	MOV @R0,#00H	MOV @R0,#00H	0069	B000
006B	18	=1673	INC R0	INC R0	006B	18
006C	E969	=1674	DJNZ K1,MAINB	DJNZ K1,MAINB	006C	E969
006E	14EC	=1675	CALL INPKEY	CALL INPKEY	006E	14EC
		=1676 ;				
		=1677 ;	WHILE KEY<MEM(OPTION+TYPE)[6-0] DO			
		=1678 ;	IF MEM(OPTION+TYPE)[7]=1 GOTO MAIND1			
		=1679 ;	TYPE:=TYPE+1			
		=1680 ;	ENDWHILE			
		=1681 ;				
		=1682	MMOV ITMP,OPTION	MMOV ITMP,OPTION		
0070	B939	=1690+	MOV R1,#OPTION	MOV R1,#OPTION	0070	B939
0072	F1	=1699+	MOV A,@R1	MOV A,@R1	0072	F1
0073	0C	=1712+	MOV ITMP,A	MOV ITMP,A	0073	0C
0074	1C	=1715	INC ITMP	INC ITMP	0074	1C
		=1716 MAINC1:	MMOV A,ITMP	MMOV A,ITMP		
0075	FC	=1732+	MOV A,ITMP	MOV A,ITMP	0075	FC
0076	E3	=1736	MOV P3 A,0A	MOV P3 A,0A	0076	E3
0077	97	=1737	CLR C	CLR C	0077	97
0078	F7	=1738	RLC A	RLC A	0078	F7
0079	77	=1739	RR A ; STRIP BIT SEVEN INTO CARRY	RR A	0079	77
007A	DE	=1740	XRL A,KEY	XRL A,KEY	007A	DE
007B	C693	=1741	JZ MAIND	JZ MAIND	007B	C693
007D	F687	=1742	JC MAIND1	JC MAIND1	007D	F687
		=1743	MINC TYPE	MINC TYPE		
007F	B937	=1748+	MOV K1,#TYPE	MOV K1,#TYPE	007F	B937
0081	F1	=1749+	MOV A,@R1	MOV A,@R1	0081	F1
0082	17	=1753+	INC A	INC A	0082	17
0083	R1	=1758+	MOV @R1,A	MOV @R1,A	0083	R1
0084	1C	=1761	INC ITMP	INC ITMP	0084	1C
0085	0475	=1762	JMP MAINC1	JMP MAINC1	0085	0475
		=1763 ;				
		=1764 ;	MODIFIER NOT FOUND SO RESET TYPE INDEX TO DEFAULT CASE (ZERO).			
		=1765 ;				
		=1766 MAIND1:	MMOV TYPE,ZERO	MMOV TYPE,ZERO		
0087	B937	=1777+	MOV R1,#TYPE	MOV R1,#TYPE	0087	B937
0089	B100	=1778+	MOV @R1,#ZERO	MOV @R1,#ZERO	0089	B100
		=1782	MMOV A,OPTION	MMOV A,OPTION		

LOC	OBJ	LINE	SOURCE STATEMENT	THIRNETH2 303RUC2	LOC	OBJ
0088	B939	=1791+	MOV R1, #OPTION	THIRNETH2 303RUC2	0088	B939
008D	F1	=1792+	MOV A, @R1	THIRNETH2 303RUC2	008D	F1
008E	E3	=1796	MOV R3, A, @R1	THIRNETH2 303RUC2	008E	E3
008F	3404	=1797	CALL OUTMSG	THIRNETH2 303RUC2	008F	3404
0091	049E	=1798	JMP MAINB0	THIRNETH2 303RUC2	0091	049E
		=1799 ;		THIRNETH2 303RUC2		
		=1800 ;	CALL OUTPUT_MESSAGE(MODIFIER)	THIRNETH2 303RUC2		
		=1801 MAIND:	MOV A, OPTION	THIRNETH2 303RUC2		
0093	B939	=1810+	MOV R1, #OPTION	THIRNETH2 303RUC2	0093	B939
0095	F1	=1811+	MOV A, @R1	THIRNETH2 303RUC2	0095	F1
0096	E3	=1815	MOV R3, A, @R1	THIRNETH2 303RUC2	0096	E3
		=1816	MADD A, TYPE	THIRNETH2 303RUC2		
0097	B937	=1822+	MOV R1, #TYPE	THIRNETH2 303RUC2	0097	B937
0099	G1	=1823+	ADD A, @R1	THIRNETH2 303RUC2	0099	G1
009A	3404	=1827	CALL OUTMSG	THIRNETH2 303RUC2	009A	3404
009C	14EC	=1828	CALL INFKEY	THIRNETH2 303RUC2	009C	14EC
		=1829 ;		THIRNETH2 303RUC2		
009E	BC00	=1830 MAINB0:	MOV ITMP, #0	THIRNETH2 303RUC2	009E	BC00
00A0	2330	=1831 MAINB1:	MOV A, #SMILO	THIRNETH2 303RUC2	00A0	2330
00A2	GC	=1832	ADD A, ITMP	THIRNETH2 303RUC2	00A2	GC
00A3	GC	=1833	ADD A, ITMP	THIRNETH2 303RUC2	00A3	GC
00A4	A8	=1834	MOV R0, A	THIRNETH2 303RUC2	00A4	A8
00A5	14C0	=1835	CALL INFADR	THIRNETH2 303RUC2	00A5	14C0
00A7	FGBA	=1836	JC CMDINT	THIRNETH2 303RUC2	00A7	FGBA
00A9	1C	=1837	INC ITMP	THIRNETH2 303RUC2	00A9	1C
00AA	B938	=1838	MOV R1, #NUMCON	THIRNETH2 303RUC2	00AA	B938
00AC	F1	=1839	MOV A, @R1	THIRNETH2 303RUC2	00AC	F1
00AD	07	=1840	DEC A	THIRNETH2 303RUC2	00AD	07
00AE	A1	=1841	MOV @R1, A	THIRNETH2 303RUC2	00AE	A1
00AF	CGBA	=1842	JZ CMDINT	THIRNETH2 303RUC2	00AF	CGBA
00B1	FB	=1843	MOV A, KEY	THIRNETH2 303RUC2	00B1	FB
00B2	D313	=1844	XRL A, #KEYEND	THIRNETH2 303RUC2	00B2	D313
00B4	CGBA	=1845	JZ CMDINT	THIRNETH2 303RUC2	00B4	CGBA
00B6	14EC	=1846	CALL INFKEY	THIRNETH2 303RUC2	00B6	14EC
00B8	04A0	=1847	JMP MAINB1	THIRNETH2 303RUC2	00B8	04A0
		=1848 ;		THIRNETH2 303RUC2		
		=1849 ;	CMDINT ENTER THE COMMAND PROCESSOR WITH:	THIRNETH2 303RUC2		
		=1850 ;	BASE_CODE=THE MAIN COMMAND TYPE	THIRNETH2 303RUC2		
		=1851 ;	TYPE=SUBCOMMAND TYPE	THIRNETH2 303RUC2		
		=1852 ;	PARAMETER(1)=FIRST ADDRESS	THIRNETH2 303RUC2		
		=1853 ;	PARAMETER(2)=SECOND ADDRESS	THIRNETH2 303RUC2		
		=1854 ;	PARAMETER(3)=DATA	THIRNETH2 303RUC2		
00BA	4400	=1855 CMDINT:	JMP IMPLM	THIRNETH2 303RUC2	00BA	4400
		=1856 ;		THIRNETH2 303RUC2		
		=1857 ;	MERROR ERROR ENCOUNTERED IN MAIN PARSING ROUTINE.	THIRNETH2 303RUC2		
00BC	BA01	=1858 MERROR:	MOV LDATA, #1	THIRNETH2 303RUC2	00BC	BA01
00BE	249A	=1859	JMP PERROR	THIRNETH2 303RUC2	00BE	249A
		=1860	SIZECHK	THIRNETH2 303RUC2		
0097		=1863+ SIZE SET 151		THIRNETH2 303RUC2	0097	
		=1864+;		THIRNETH2 303RUC2		
		=1865+;	*****	THIRNETH2 303RUC2		
		=1874 \$EJECT		THIRNETH2 303RUC2		

LOC	OBJ	LINE	SOURCE STATEMENT	THIRDTAT2	SCRU02	LOC	OBJ
		=1875	DATABLK \$0				
0323		=1880+ =1884 ;	ORG 003				
		=1885 ;	*****				
		=1886 ;	*****				
		=1887 ;	*****				
		=1888 ;	*****				
		=1889 ;	*****				
		=1890 ;	*****				
		=1891 ;	THE CTAB TABLE CONTAINS <COMSIZ> ENTRIES FOR EACH COMMAND. THE MEANING				
		=1892 ;	OF THE ENTRIES IS AS FOLLOWS:				
		=1893 ;					
		=1894 ;	ENTRY 0. COMMAND KEY TO INITIATE				
		=1895 ;	ENTRY 1. POINTER TO THE LIST OF OPTIONS APPLICABLE TO THIS COMMAND				
		=1896 ;	ENTRY 2. NUMBER OF NUMERIC PARAMETERS REQUIRED BY THE COMMAND				
		=1897 ;					
0023		=1898 CTAB	EQU \$ AND OFFH				
0003		=1899 COMSIZ	EQU 3				
		=1900 ;					
0323 1F		=1901	DB KEYMOD, LOW OPTAB1, 1 ; EXAM				
0324 3F		=					
0325 01		=					
0326 1E		=1902	DB KEYGO, LOW OPTAB3, 1 ; GO				
0327 49		=					
0328 01		=					
0329 10		=1903	DB KEYFIL, LOW OPTAB1, 3 ; FILL				
032A 3F		=					
032B 03		=					
032C 1C		=1904	DB KEYLST, LOW OPTAB1, 2 ; DUMP				
032D 3F		=					
032E 02		=					
032F 18		=1905	DB KEYREC, LOW OPTAB1, 2 ; RECORD				
0330 3F		=					
0331 02		=					
0332 14		=1906	DB KEYREL, LOW OPTAB1, 0 ; RELOAD				
0333 3F		=					
0334 00		=					
0335 00		=1907	DB KSE1B, LOW OPTAB2, 1 ; SETBRK				
0336 46		=					
0337 01		=					
0338 0C		=1908	DB KCLR8, LOW OPTAB2, 1 ; CLRBRK				
0339 46		=					
033A 01		=					
033B 1D		=1909	DB KGORES, LOW OPTAB3, 0 ; GO FROM RESET STATE				
033C 49		=					
033D 00		=					
033E FF		=1910	DB OFFH ; ESCOP				
		=1911 ;					
		=1912 \$EJECT					

LOC	OBJ	LINE	SOURCE STATEMENT	TRANSLATED STATEMENT	LOC	OBJ
0129	69	=2051	DB LOW(DINTRG) ; DATA TYPE MODIFIER 3 RESPONSE ADDRESS	DB LOW(DINTRG)	0129	69
012A	65	=2052	DB LOW(DPRBRK) ; DATA TYPE MODIFIER 4 RESPONSE ADDRESS	DB LOW(DPRBRK)	012A	65
012B	67	=2053	DB LOW(DDABRK) ; DATA TYPE MODIFIER 5 RESPONSE ADDRESS	DB LOW(DDABRK)	012B	67
002C		=2054	SYNOC EQU LOW \$	SYNOC EQU LOW \$	002C	
012C	6B	=2055	DB LOW(DNDBRK) ; EXECUTION MODE MODIFIER 0	DB LOW(DNDBRK)	012C	6B
012D	6D	=2056	DB LOW(DMBRK) ; EXECUTION MODE MODIFIER 1	DB LOW(DMBRK)	012D	6D
012E	6F	=2057	DB LOW(DSS) ; EXECUTION MODE MODIFIER 2	DB LOW(DSS)	012E	6F
012F	72	=2058	DB LOW(DPA) ; EXECUTION MODE MODIFIER 3	DB LOW(DPA)	012F	72
0130	75	=2059	DB LOW(DTR) ; EXECUTION MODE MODIFIER 4	DB LOW(DTR)	0130	75
		=2060	UTILITY OUTPUT MESSAGES			
		=2061				
		=2062				
		=2063	ERROR:			
0131	79	=2064	DB 01111001B ; "E"	DB 01111001B	0131	79
0132	50	=2065	DB 01010000B ; "R"	DB 01010000B	0132	50
0133	50	=2066	DB 01010000B ; "R"	DB 01010000B	0133	50
0134	5C	=2067	DB 01011100B ; "O"	DB 01011100B	0134	5C
0135	50	=2068	DB 01010000B ; "R"	DB 01010000B	0135	50
0136	C0	=2069	DB 11000000B ; "."	DB 11000000B	0136	C0
		=2070	DSGNON:			
0137	00	=2071	DB 00000000B ; " "	DB 00000000B	0137	00
0138	76	=2072	DB 01110110B ; "H"	DB 01110110B	0138	76
0139	6D	=2073	DB 01101101B ; "S"	DB 01101101B	0139	6D
013A	79	=2074	DB 01111001B ; "E"	DB 01111001B	013A	79
013B	40	=2075	DB 01000000B ; "-"	DB 01000000B	013B	40
013C	66	=2076	DB 01100110B ; "4"	DB 01100110B	013C	66
013D	E7	=2077	DB 11100111B ; "9" (TH)	DB 11100111B	013D	E7
		=2078	DRUN:			
013E	00	=2079	DB 00000000B ; "I"	DB 00000000B	013E	00
013F	40	=2080	DB 01000000B ; "-"	DB 01000000B	013F	40
0140	50	=2081	DB 01010000B ; "R"	DB 01010000B	0140	50
0141	1C	=2082	DB 00011100B ; "U"	DB 00011100B	0141	1C
0142	54	=2083	DB 01010100B ; "N"	DB 01010100B	0142	54
0143	C0	=2084	DB 11000000B ; "."	DB 11000000B	0143	C0
		=2085	DEPNT:			
0144	73	=2086	DB 01110011B ; "P"	DB 01110011B	0144	73
0145	B9	=2087	DB 10111001B ; "C"	DB 10111001B	0145	B9
		=2088	\$JECT			

LOC	OBJ	LINE	SOURCE STATEMENT	INSTRUC	33002	LOC	OBJ
		=2089 ;					
		=2090 ;	PRIMARY COMMAND RESPONSE STRING PATTERNS				
		=2091 ;					
		=2092 DMOD:					
0146	79	=2093	DB 01111001B, 00111001B, 11110100B ; "ECH."	DB			
0147	79	=					
0148	F4	=					
		=2094 DGO:					
0149	3D	=2095	DB 00111101B, 11011100B ; "GO."	DB			
014A	DC	=					
		=2096 DFILL:					
014B	71	=2097	DB 01110001B, 00110000B, 10111000B ; "FIL."	DB			
014C	30	=					
014D	B8	=					
		=2098 DLST:					
014E	38	=2099	DB 00111000B, 01101101B, 11111000B ; "LST."	DB			
014F	6D	=					
0150	F8	=					
		=2100 DREC:					
0151	3E	=2101	DB 00111110B, 01110011B, 10111000B ; "UPL."	DB			
0152	73	=					
0153	B8	=					
		=2102 DREL:					
0154	5E	=2103	DB 01011110B, 01010100B, 10111000B ; "DNL."	DB			
0155	54	=					
0156	B8	=					
		=2104 DSE:					
0157	6D	=2105	DB 01101101B, 01111000B, 11111000B ; "STB."	DB			
0158	78	=					
0159	FC	=					
		=2106 DCD:					
015A	39	=2107	DB 00111001B, 00111000B, 11111000B ; "CLD."	DB			
015B	38	=					
015C	FC	=					
		=2108 DGR:					
015D	3D	=2109	DB 00111101B, 11010000B ; "GR."	DB			
015E	D0	=					
		=2110 \$EJECT					

LOC	OBJ	LINE	SOURCE STATEMENT	THREATS 33A02	LOC	OBJ
		=2111 ;				
		=2112 ;	MEMORY SPACE MODIFIER OPTION RESPONSE STRINGS			
		=2113 ;				
		=2114 DFRMEN:				
015F 73		=2115	DB 01110011B, 11010000B ; "PR "			
0160 D0		=				
		=2116 DDAMEN:				
0161 5E		=2117	DB 01011110B, 11110111B ; "DN "			
0162 F7		=				
		=2118 DRM:				
0163 50		=2119	DB 01010000B, 10111101B ; "RG "			
0164 BD		=				
		=2120 DFRBKK:				
0165 73		=2121	DB 01110011B, 11111100B ; "PD "			
0166 FC		=				
		=2122 DDABKK:				
0167 5E		=2123	DB 01011110B, 11111100B ; "DE "			
0168 FC		=				
		=2124 DINTRG:				
0169 76		=2125	DB 01110110B, 11010000B ; "HR "			
016A D0		=				
		=2126 ;				
		=2127 ;	RESPONSE MESSAGES FOR GU CONDITION MODIFIERS.			
		=2128 ;				
		=2129 DNOBRK:				
016B 54		=2130	DB 01010100B, 11111100B ; "ND "			
016C FC		=				
		=2131 DMBRK:				
016D 7C		=2132	DB 01111100B, 11010000B ; "BR "			
016E D0		=				
		=2133 DSS:				
016F 6D		=2134	DB 01101101B, 01101101B, 11111100B ; "SST "			
0170 6D		=				
0171 F8		=				
		=2135 DPA:				
0172 77		=2136	DB 01110111B, 01111100B, 11010000B ; "ABR "			
0173 7C		=				
0174 D0		=				
		=2137 DTR:				
0175 77		=2138	DB 01110111B, 01101101B, 11111100B ; "AST "			
0176 6D		=				
0177 F8		=				
		=2139 ;				
		=2140	SIZECHK			
0078		=2143+ SIZE SET 120				
		=2144+;				
		=2145+; *****				
		=2154 \$EJECT				

LOC	OBJ	LINE	SOURCE STATEMENT	THIRDTIME 334002	LOC	OBJ
		=2155	CODECLK 45			
00C0		=2160+	ORG 192			
		=2164 ;	INPADR INPUT DATA INTO TWO-BYTE PARAMETER BUFFER INDICATED BY R0.			
		=2165 ;	RECEIVE NUMERIC KEYS FROM KEYBOARD UNTIL ', ' OR '. '.			
		=2166 ;	SHIFT INTO ADDRESS BUFFER;			
		=2167 ;	RE-WRITE DISPLAY.			
		=2168 ;	IF NUMBER OF CONSTANTS NEEDED IS ZERO, NO NEW PARAMETERS ARE ALLOWED.			
		=2169 ;				
00C0 97		=2170	INPADR: CLR C			
00C1 A7		=2171	CPL C			
		=2172	MNOV A, NUMCON			
00C2 B930		=2181+	MOV RL, #NUMCON			
00C4 F1		=2182+	MOV A, R1			
00C5 C6D7		=2186	JZ ELSIF1			
00C7 FB		=2187	INPAD1: MOV A, KEY			
00C8 92D7		=2188	JB4 ELSIF1			
00CA 20		=2189	XCH A, R0			
00CB 47		=2190	SWAP A			
00CC 20		=2191	XCH A, R0			
00CD 30		=2192	XCHD A, R0			
00CE 18		=2193	INC R0			
00CF 30		=2194	XCHD A, R0			
00D0 3478		=2195	CALL UPDADR			
00D2 14EC		=2196	CALL INPKEY			
00D4 97		=2197	CLR C			
00D5 04C7		=2198	JMP INPAD1			
		=2199 ;				
		=2200 ;	ELSIF1 IF KEY=', ' OR '. ' THEN RETURN.			
		=2201 ;				
00D7 FB		=2202	ELSIF1: MOV A, KEY			
00D8 D312		=2203	XRL A, #KEYNXT			
00DA C6E5		=2204	JZ ELSIF2			
00DC FB		=2205	MOV A, KEY			
00DD D313		=2206	XRL A, #KEYEND			
00DF C6E5		=2207	JZ ELSIF2			
		=2208 ;				
		=2209 ;	ELSE GOTO PERROR.			
		=2210 ;				
00E1 BA02		=2211	MOV LDATA, #2			
00E3 249A		=2212	JMP PERROR			
00E5 D846		=2213	ELSIF2: MOV R0, #SEGMAP			
00E7 B903		=2214	MOV RL, #3			
00E9 B4F5		=2215	CALL DGLANK			
00EB 83		=2216	RET			
		=2217	SIZECHK			
002C		=2220+	SIZE SET 44			
		=2221+;				
		=2222+;	*****			
		=2231	\$EJECT			

LOC	OBJ	LINE	SOURCE STATEMENT	THIRTEENTH SOURCE STATEMENT	LOC
		=2298	CODEBLK 35		
0198		=2300+	ORG 400		
		=2312 ;	ERROR: REPEAT		
		=2313 ;	OUTPUT_MESSAGE(PERROR_PROMPT)		
		=2314 ;	OUTPUT(LDATA)		
		=2315 ;	CALL INPUT_BYTE(KEY)		
		=2316 ;	UNTIL KEY='CLEAR/PREVIOUS'		
0190 0A04		=2317 ERROR: MOV	LDATA, #4		
019A BF02		=2318 ERROR: MOV	XPCODE, #2		
019C 74D1		=2319	CALL XPTST		
019E 27		=2320	CLR A		
019F D7		=2321	MOV PSW, A		
01A0 FB		=2322	MOV A, KEY		
01A1 D317		=2323	XRL A, #KEYCLR		
01A3 C6D6		=2324	JZ ERROR2		
01A5 27		=2325	CLR A		
01A6 3400		=2326	CALL OUTUTL		
01A8 FA		=2327	MOV A, LDATA		
01A9 D4D3		=2328	CALL DSPACC		
		=2329	MMOV KBDDUF, NEG1		
01AB B93B		=2340+	MOV R1, #KDDBUF		
01AD B11F		=2341+	MOV @R1, #NEG1		
01AF 14EC		=2345	CALL INPKEY		
01B1 FD		=2346	MOV A, KEY		
01B2 D313		=2347	XRL A, #KEYEND		
01B4 9698		=2348	JNZ ERROR		
01B6 0429		=2349 ERROR2: JMP	MAIN		
		=2350	SIZELCK		
0020		=2353+ SIZE SET 32			
		=2354+			
		=2355+ ;	*****		
		=2364 ;			
		=2365	CODEBLK 80		
0200		=2380+	ORG 512		
		=2384 ;	IMPLEM IMPLEMENT COMMAND		
0200 2306		=2385 IMPLM: MOV	A, #LOW(JMPTBL)		
		=2386	MADD A, BCODE		
0202 B936		=2392+	MOV R1, #BCODE		
0204 61		=2393+	ADD A, @R1		
0205 B3		=2397	JMP @A		
		=2398 ;			
		=2399 JMPTBL:			
0206 0F		=2400	DB LOW(JTOMOD)		
0207 20		=2401	DB LOW(JTGO)		
0208 22		=2402	DB LOW(JTOFIL)		
0209 1A		=2403	DB LOW(JTOLST)		
020A 11		=2404	DB LOW(JTOREC)		
020B 16		=2405	DB LOW(JTOREL)		
020C 2C		=2406	DB LOW(COMSER)		
020D 28		=2407	DB LOW(COMCER)		
020E 26		=2408	DB LOW(JGORES)		
		=2409 ;			
020F 444F		=2410 JTOMOD: JMP	EXAMIN		
		=2411 ;			
0211 85		=2412 JTOREC: CLR	F0 ; F0=0 ==> HEX FORMAT DATA DUMP		

```

0212 B472 =2413 CALL HFILED
0214 0429 =2414 JMP MAIN
          =2415 ;
0216 5497 =2416 JTOREL CALL HRECIN
0218 0429 =2417 JMP MAIN
          =2418 ;
021A 05 =2419 JTOIST: CLR F0
021B 95 =2420 CPL F0
021C B472 =2421 CALL HFILED
021E 0429 =2422 JMP MAIN
          =2423 ;
0220 8400 =2424 JTOGO: JMP EPRUN
          =2425 ;
0222 54E5 =2426 JTOFIL: CALL COMFIL
0224 0429 =2427 JMP MAIN
          =2428 ;
0226 8461 =2429 JTORES: JMP COMGOR
          =2430 ;
          =2431 ; COMCBR COMMAND TO CLEAR BREAKPOINTS
0228 B800 =2432 COMCBR: MOV LDATA, #0
022A 442E =2433 JMP BRKFIL
          =2434 ;
          =2435 ; COMSBR COMMAND TO SET BREAKPOINTS
022C B801 =2436 COMSBR: MOV LDATA, #1
022E 2304 =2437 BRKFIL: MOV A, #4
          =2438 MADD TYPE, A
0230 B937 =2448+ MOV R1, #TYPE
0232 61 =2449+ ADD A, @R1
0233 A1 =2455+ MOV @R1, A
0234 F400 =2459 BRKNXT: CALL LSTORE
0236 FB =2460 MOV A, KEY
0237 D313 =2461 XRL A, #KEYEND
0239 C64D =2462 JZ BRKEND
023B 14EC =2463 CALL INPKEY
          =2464 MMOV NUMCON, PLUS1
023D B938 =2475+ MOV R1, #NUMCON
023F D101 =2476+ MOV @R1, #PLUS1
0241 B830 =2480 MOV R0, #SMALO
0243 B000 =2481 MOV @R0, #0
          =2482 MMOV SMH1, ZERO
0245 B931 =2493+ MOV R1, #SMH1
0247 B100 =2494+ MOV @R1, #ZERO
0249 14C0 =2498 CALL INPADR
024B E634 =2499 JNC BRKNXT
024D 0429 =2500 BRKEND: JMP MAIN
          =2501 SIZECHK
004F =2504+ SIZE SET 79
          =2505+;
          =2506+; *****
          =2515 $EJECT

```

LOC	OBJ	LINE	SOURCE STATEMENT	THIRDTIME	THIRDSOURCE	THIRDSIZE	THIRDSLOC
		=2516	CODEBLK 75				
024F		=2531	ORG 591	(CHECK/END) A	ORG	591	1100
		=2535	EXAMIN EXAMINE/MODIFY MEMORY COMMAND.	SWAPC	SWC	10	1100
		=2536	DISPLAYS MEMORY ADDRESS SPACE OPTION, ADDRESS VALUE, AND CURRENT DATA.				1100
		=2537	READS KEYBOARD AND INTERPRETS RESPONSE.				1100
		=2538		VEN A	VEN	SWAPC	10
		=2539	OUTPUT_MESSAGE((MEMORY_SPACE_OPTION)<SMA>/'<DATA_BYTE>')				1100
024F 85		=2540	EXAMIN: CLR F0	ENR0	SWC	10	1100
		=2541	EXAM0: MOV A, TYPE	MOV0	SWC	10	1100
0250 0937		=2550	MOV R1, #TYPE	MOV0	SWC	10	1100
0252 F1		=2551	MOV A, R1	MOV A	SWC	10	1100
0253 0326		=2555	ADD A, #STRMEM ; OFFSET FOR FIRST MEMORY TYPE STRING	ADD	SWC	10	1100
0255 3402		=2556	CALL OUTCLR	CALL	SWC	10	1100
0257 0831		=2557	MOV R0, #SMALO+1	MOV	SWC	10	1100
0259 347C		=2558	CALL UPDAD1	CALL	SWC	10	1100
025B 2348		=2559	MOV A, #01001000B ; '='	MOV	SWC	10	1100
025D 0408		=2560	CALL WDISP	CALL	SWC	10	1100
025F 14FC		=2561	CALL LFETCH	CALL	SWC	10	1100
0261 FA		=2562	MOV A, LDATA	MOV	SWC	10	1100
0262 47		=2563	SWAP A	SWAP	SWC	10	1100
0263 0403		=2564	CALL DSPACC	CALL	SWC	10	1100
0265 FA		=2565	MOV A, LDATA	MOV	SWC	10	1100
0266 0403		=2566	CALL DSPACC	CALL	SWC	10	1100
		=2567					
		=2568					
		=2569	INPUT_KEY(KEY)				
		=2570	IF (KEY IS NOT NUMERIC)				
		=2571	IF (KEY=KEYEND) GO TO PARSER				
		=2572	ELSEIF (KEY=KEYNEXT)				
		=2573	INCREMENT <SMA>				
		=2574	GOTO EXAMIN				
		=2575	ELSEIF (KEY=KEYPREVIOUS)				
		=2576	DECREMENT <SMA>				
		=2577	GOTO EXAMIN				
		=2578	ELSE GOTO ERROR				
		=2579					
0260 14EC		=2580	CALL INPKEY				
		=2581	MOV A, KEY				
026A FB		=2597	MOV A, KEY				
026B 927B		=2601	JB4 EXAM1				
		=2602					
		=2603	APPEND DATA WITH <LOWNIB-<KEY>>				
		=2604	CALL LSTORE				
		=2605	GOTO EXAMIN				
		=2606					
026D FA		=2607	MOV A, LDATA				
026E 47		=2608	SWAP A				
026F 53F0		=2609	ANL A, #0F0H				
0271 0675		=2610	JF0 EXAM5				
0273 27		=2611	CLR A				
0274 95		=2612	CPL F0				
0275 6B		=2613	EXAM5: ADD A, KEY				
0276 0A		=2614	MOV LDATA, A				
0277 F400		=2615	CALL LSTORE				
0279 4450		=2616	JMP EXAM0				

LOC	OBJ	LINE	SOURCE STATEMENT	THIRDTIME2	CONJUG2	LOC	OBJ
		=2617 ;					
027B	D313	=2618 EXAM1:	XRL A, #KEYEND)				
027D	9681	=2619	JNZ EXAM2				
027F	0429	=2620	JMP MAIN				
		=2621 ;					
0281	FB	=2622 EXAM2:	MOV A, KEY				
0282	D312	=2623	XRL A, #KEYVMT				
0284	968A	=2624	JNZ EXAM3				
0286	34F2	=2625	CALL INCSMA				
0288	444F	=2626	JMP EXAMIN				
028A	FB	=2627 EXAM3:	MOV A, KEY				
028B	D317	=2628	XRL A, #KEYCLR				
028D	9693	=2629	JNZ EXAM4				
028F	54F4	=2630	CALL DECSMA				
0291	444F	=2631	JMP EXAMIN				
0293	B703	=2632 EXAM4:	MOV LDATA, #03H				
0295	249A	=2633	JMP ERROR				
		=2634	SIZECHK				
0048		=2637+ SIZE	SET 72				
		=2638+;					
		=2639+; *****					
		=2648 ;					
		=2649	CODEBLK 4				
00EC		=2654+	ORG 236				
00EC	D4C2	=2658 INPKV:	CALL KBDIN ; RETURNS KEY DEPRESSION IN A				
00EE	AD	=2659	MOV KEY, A				
00EF	83	=2660	RET				
		=2661	SIZECHK				
0004		=2664+ SIZE	SET 4				
		=2665+;					
		=2666+; *****					
		=2675 \$EJECT					

LOC	OBJ	LINE	SOURCE STATEMENT	THIRTEENTH	THIRTEENTH	THIRTEENTH
		2676 \$	INCLUDE(:F0:GOCONS.MOD)			
		=2677	CODEBLK 210			
0400		=2697+	ORG 1024			
		=2701	EPRUN RUN EMULATION MODE.			
		=2702 ;	RELOAD EP WITH SYSTEM STATUS AND RELEASE			
		=2703 ;	SEQUENCE IS AS FOLLOWS:			
		=2704 ;	IF COMMAND WAS TERMINATED BY THE 'NEXT' KEY:			
		=2705 ;	STORE SMA INTO EP PC;			
		=2706 ;	STORE EP PC INTO TOP-OF-STACK (RELATIVE 10 EP PSM);			
		=2707 ;	PASS EP R0;			
		=2708 ;	PASS EP PSM;			
		=2709 ;	PASS EP TIMER;			
		=2710 ;	PASS EP ACCUMULATOR;			
		=2711 ;				
0400	2302	=2712 EPRUN:	MOV A, #2			
0402	3400	=2713	CALL OUTUTL			
		=2714	MMOV A, NUMCON			
0404	B938	=2723+	MOV R1, #NUMCON			
0406	F1	=2724+	MOV A, R1			
0407	9615	=2728	JNZ EPCONT			
		=2729	MMOV EPPCLO, SMAILO			
0409	B930	=2745+	MOV R1, #SMAILO			
040B	F1	=2746+	MOV A, R1			
040C	B924	=2752+	MOV R1, #EPPCLO			
040E	A1	=2753+	MOV R1, A			
		=2756	MMOV EPCHI, SMAHI			
040F	B931	=2772+	MOV R1, #SMAHI			
0411	F1	=2773+	MOV A, R1			
0412	B925	=2779+	MOV R1, #EPPCHI			
0414	A1	=2780+	MOV R1, A			
0415	FB	=2783 EPCONT:	MOV A, KEY			
0416	D312	=2784	XRL A, #KEYNXT			
0418	C61F	=2785	JZ EPCON1			
041A	2301	=2786	MOV A, #01H ; STACK ONE LEVEL DEEP TO HOLD USER STARTING ADDRESS			
		=2787	MMOV EPPSW, A			
041C	B921	=2800+	MOV R1, #EPPSW			
041E	A1	=2801+	MOV R1, A			
		=2805 EPCON1:	MMOV LDATA, EPPCLO			
041F	B924	=2821+	MOV R1, #EPPCLO			
0421	F1	=2822+	MOV A, R1			
0422	AA	=2835+	MOV LDATA, A			
		=2838	MMOV A, EPPSW			
0423	B921	=2847+	MOV R1, #EPPSW			
0425	F1	=2848+	MOV A, R1			
0426	07	=2852	DEC A			
0427	5307	=2853	ANL A, #07H			
0429	E7	=2854	RL A			
042A	0308	=2855	ADD A, #00H			
		=2856	MMOV SMAILO, A			
042C	B930	=2869+	MOV R1, #SMAILO			
042E	A1	=2870+	MOV R1, A			
042F	F4C3	=2874	CALL EPSTOR			
		=2875	INCR SMAILO			
0431	B930	=2880+	MOV R1, #SMAILO			
0433	F1	=2881+	MOV A, R1			

LOC	OBJ	LINE	SOURCE STATEMENT	OBJECT STATEMENT	LOC	OBJ
0434	17	=2885+	INC A	INC A	0434	17
0435	R1	=2890+	MOV @R1, A	MOV @R1, A	0435	R1
		=2893	MNOV A, EPSPW	MNOV A, EPSPW		
0436	B921	=2902+	MOV R1, #EPSPW	MOV R1, #EPSPW	0436	B921
0438	F1	=2903+	MOV A, @R1	MOV A, @R1	0438	F1
0439	53F0	=2907	ANL A, #0F0H	ANL A, #0F0H	0439	53F0
		=2908	MORL A, EPPCHI	MORL A, EPPCHI		
0438	B925	=2914+	MOV R1, #EPPCHI	MOV R1, #EPPCHI	0438	B925
043D	41	=2915+	ORL A, @R1	ORL A, @R1	043D	41
043E	0A	=2919	MOV LDATH, A	MOV LDATH, A	043E	0A
043F	F4C3	=2920	CALL EPSTOR	CALL EPSTOR	043F	F4C3
0441	B8D1	=2921 EPCNT:	MOV R0, #LOW(OV2BAS+OVSIZE)	MOV R0, #LOW(OV2BAS+OVSIZE)	0441	B8D1
0443	746A	=2922	CALL OVLORD	CALL OVLORD	0443	746A
		=2923	MNOV A, EPR0	MNOV A, EPR0		
0445	B923	=2932+	MOV R1, #EPR0	MOV R1, #EPR0	0445	B923
0447	F1	=2933+	MOV A, @R1	MOV A, @R1	0447	F1
0448	F4D0	=2937	CALL EPPASS	CALL EPPASS	0448	F4D0
		=2938	MNOV A, EPSPW	MNOV A, EPSPW		
044A	B921	=2947+	MOV R1, #EPSPW	MOV R1, #EPSPW	044A	B921
044C	F1	=2948+	MOV A, @R1	MOV A, @R1	044C	F1
044D	F4D0	=2952	CALL EPPASS	CALL EPPASS	044D	F4D0
		=2953	MNOV A, EPTIMR	MNOV A, EPTIMR		
044F	B922	=2962+	MOV R1, #EPTIMR	MOV R1, #EPTIMR	044F	B922
0451	F1	=2963+	MOV A, @R1	MOV A, @R1	0451	F1
0452	F4D0	=2967	CALL EPPASS	CALL EPPASS	0452	F4D0
		=2968	MNOV A, EPACC	MNOV A, EPACC		
0454	B920	=2977+	MOV R1, #EPACC	MOV R1, #EPACC	0454	B920
0456	F1	=2978+	MOV A, @R1	MOV A, @R1	0456	F1
0457	F4D0	=2982	CALL EPPASS	CALL EPPASS	0457	F4D0
0459	8903	=2983	ORL P1, #00000011B	ORL P1, #00000011B	0459	8903
045B	F4D8	=2984	CALL EPSTEP	CALL EPSTEP	045B	F4D8
045D	745A	=2985	CALL OVSWAP	CALL OVSWAP	045D	745A
045F	846B	=2986	JMP CGO	JMP CGO	045F	846B
		=2987 ;	CONGOR GO FROM RESET COMMAND	CONGOR GO FROM RESET COMMAND		
		=2989 ;	RESET PROCESSOR	RESET PROCESSOR		
		=2990 ;	RELOAD LOW ORDER PROGRAM BYTES INTO PROGRAM MEMORY	RELOAD LOW ORDER PROGRAM BYTES INTO PROGRAM MEMORY		
		=2991 ;				
0461	2302	=2992 CONGOR:	MOV A, #2	MOV A, #2	0461	2302
0463	3400	=2993	CALL OUTUTL	CALL OUTUTL	0463	3400
0465	8910	=2994	ORL P1, #EPRSET	ORL P1, #EPRSET	0465	8910
0467	745A	=2995	CALL OVSWAP	CALL OVSWAP	0467	745A
0469	99EF	=2996	ANL P1, #(NOT EPRSET)	ANL P1, #(NOT EPRSET)	0469	99EF
		=2997 ;				
		=2998 ;				
		=2999 ; CGO	SET UP BREAK LOGIC FOR APPROPRIATE BREAK CONDITIONS,	SET UP BREAK LOGIC FOR APPROPRIATE BREAK CONDITIONS,		
		=3000 ;	DEPENDING ON CONTENTS OF 'TYPE'.	DEPENDING ON CONTENTS OF 'TYPE'.		
		=3001 ;				
		=3002 CGO:	MNOV A, TYPE	MNOV A, TYPE		
046B	B937	=3011+	MOV R1, #TYPE	MOV R1, #TYPE	046B	B937
046D	F1	=3012+	MOV A, @R1	MOV A, @R1	046D	F1
046E	0371	=3016	ADD A, #LOW GOTBL	ADD A, #LOW GOTBL	046E	0371
0470	B3	=3017	JNPP @A	JNPP @A	0470	B3
		=3018 ;				
0471	7C	=3019 GOTBL:	DB LOW(CGONE)	DB LOW(CGONE)	0471	7C

LOC	OBJ	LINE	SOURCE STATEMENT	DISASSEMBLY	LOC	OBJ
0472	76	=3020	DB LOW(CGOMB)	DB LOW(CGOMB)	0472	76
0473	80	=3021	DB LOW(CGOS5)	DB LOW(CGOS5)	0473	80
0474	76	=3022	DB LOW(CGOPAT)	DB LOW(CGOPAT)	0474	76
0475	80	=3023	DB LOW(CGOTRA)	DB LOW(CGOTRA)	0475	80
		=3024 ;				
		=3025 CGOPAT:				
0476	99FD	=3026 CGOMB: ANL	P1, #NOT 000000100	ANL P1, #NOT 000000100	0476	99FD
0478	8501	=3027	ORL P1, #00000001B	ORL P1, #00000001B	0478	8501
047A	8482	=3028	JMP EPRUN4	JMP EPRUN4	047A	8482
		=3029 ;				
047C	99FC	=3030 CGOMB: ANL	P1, #NOT 00000011B	ANL P1, #NOT 00000011B	047C	99FC
047E	8482	=3031	JMP EPRUN4	JMP EPRUN4	047E	8482
		=3032 ;				
		=3033 CGOTRA:				
0480	8503	=3034 CGOS5: ORL	P1, #00000011B	ORL P1, #00000011B	0480	8503
		=3035 ;				
		=3036 ; EPRUN4 SET UP CONTROL LOGIC TO RUN USER'S PROGRAM.				
		=3037 ; RELEASE PROCESSOR TO RUN.				
		=3038 ;				
0482	8A20	=3039 EPRUN4: ORL	P2, #00100000B ; DISABLE EP LINK REFERENCES.	ORL P2, #00100000B ; DISABLE EP LINK REFERENCES.	0482	8A20
0484	9A1F	=3040	ANL P2, #NOT 00010000B ; SET ALL REFERENCES TO RAM ARRAY.	ANL P2, #NOT 00010000B ; SET ALL REFERENCES TO RAM ARRAY.	0484	9A1F
0486	99DF	=3041	ANL P1, #NOT MODOUT	ANL P1, #NOT MODOUT	0486	99DF
0488	F4F4	=3042	CALL EPREL	CALL EPREL	0488	F4F4
		=3043 ;				
		=3044 ; WAIT FOR KEYSTROKE INPUT OR HARDWARE BREAK TO OCCUR.				
		=3045 ;				
048A	F4AC	=3046 EPRUN1: CALL	IOFPOL	CALL IOFPOL	048A	F4AC
048C	F4AF	=3047	CALL KBDPOL	CALL KBDPOL	048C	F4AF
048E	37	=3048	CPL A	CPL A	048E	37
048F	F295	=3049	JB7 EPRUN3	JB7 EPRUN3	048F	F295
0491	8699	=3050	JNI EPRUN2	JNI EPRUN2	0491	8699
0493	848A	=3051	JMP EPRUN1	JMP EPRUN1	0493	848A
		=3052 ;				
		=3053 ; EPRUN3 A KEYSTROKE WAS DETECTED WHILE EP WAS RUNNING.				
		=3054 ; BREAK EXECUTION.				
		=3055 ; PROCESS KEYSTROKE.				
0495	D400	=3056 EPRUN3: CALL	STSAVE	CALL STSAVE	0495	D400
0497	84B3	=3057	JMP EPRUN5	JMP EPRUN5	0497	84B3
		=3058 ;				
		=3059 ; EPRUN2 AN ENABLED BREAK CONDITION OCCURRED.				
		=3060 ; BREAK EMULATION MODE.				
		=3061 ; CONTINUE ACCORDING TO GO COMMAND TYPE.				
0499	B400	=3062 EPRUN2: CALL	STSAVE	CALL STSAVE	0499	B400
		=3063	MNOV A, TYPE	MNOV A, TYPE		
049B	B937	=3072	MOV R1, #TYPE	MOV R1, #TYPE	049B	B937
049D	F1	=3073	MOV A, @R1	MOV A, @R1	049D	F1
049E	03A1	=3077	ADD A, #LOW CNTTBL	ADD A, #LOW CNTTBL	049E	03A1
04A0	B3	=3078	JMPP @A	JMPP @A	04A0	B3
		=3079 ;				
04A1	A6	=3080 CNTTBL: DB	LOW(BRKERR)	DB LOW(BRKERR)	04A1	A6
04A2	BA	=3081	DB LOW(EPRUN3)	DB LOW(EPRUN3)	04A2	BA
04A3	BA	=3082	DB LOW(EPRUN3)	DB LOW(EPRUN3)	04A3	BA
04A4	AA	=3083	DB LOW(CNTTRA)	DB LOW(CNTTRA)	04A4	AA
04A5	AA	=3084	DB LOW(CNTTRA)	DB LOW(CNTTRA)	04A5	AA
		=3085 ;				

LOC	OBJ	LINE	SOURCE STATEMENT	THIRDTAT2 33RUC2	LOC	OBJ
		=3086 ;	BRKERR BREAKPOINT LATCH WAS SET THOUGH BREAKPOINTS NOT ENABLED			
		=3087 ;	DISPLAY HARDWARE ERROR MESSAGE			
04A6	BA0B	=3088 BRKERR:	MOV LDATA, #0EH			
04A8	249A	=3089	JMP PERROR			
		=3090 ;				
		=3091 CNTTRA:	MMOV R, DSPTIM			
04AA	D928	=3100+	MOV R1, #DSPTIM			
04AC	F1	=3101+	MOV R, R1			
04AD	94F2	=3105	CALL DELAY			
04AF	F4AF	=3106	CALL KBDPOL			
04B1	F241	=3107	JB7 EPCNT ; B7 SET INDICATES NO KEYSTROKE			
		=3108 ;				
		=3109 ;	EPRUN5 INPUT(KEY),			
		=3110 ;	IF KEY=END GO TO PARSER,			
		=3111 ;	INPUT KEY,			
		=3112 ;	IF KEY<NEXT GO TO PARSER,			
		=3113 ;	CONTINUE IN SAME MODE.			
		=3114 ;				
04B3	14EC	=3115 EPRUN5:	CALL INPKEY			
04B5	FB	=3116	MOV A, KEY			
04B6	D313	=3117	XRL A, #KEYEND			
04B8	96C7	=3118	JNZ EPRET			
04BA	14EC	=3119 EPRUN6:	CALL INPKEY			
04BC	FB	=3120	MOV A, KEY			
04BD	D312	=3121	XRL A, #KEYNEXT			
04BF	96C7	=3122	JNZ LPRET			
04C1	2302	=3123	MOV A, #2			
04C3	3400	=3124	CALL OUTUTL			
04C5	8441	=3125	JMP EPCNT			
		=3126 ;				
		=3127 ;	EPRET EXECUTION MODE IS TO BE TERMINATED.			
		=3128 ;	JUMP INTO PARSER TO INTERPRET KEY ALREADY DETECTED.			
04C7	0433	=3129 EPRET:	JMP MAIN2			
		=3130 ;				
		=3131	SIZECHK			
00C9		=3134+ SIZE	SET 201			
		=3135+;				
		=3136+; *****				
		=3145 #EJECT				

LOC	OBJ	LINE	SOURCE STATEMENT	THIRDTATZ	30R002	LOC	OBJ
		=3146	CODEBLK 115	A	JR		
0500		=3171+	ORG 1200	MOV	A, 000		
		=3175	STSAVE EP STATUS SAVE SUBROUTINE.	A, 0100	VON		
		=3176	FORCE CALL TO LOC 014H;	CALL	JR		
		=3177	SAVE EP ACC;	A, 000	VON		
		=3178	SAVE EP TIMER;	TEPR	JR		
		=3179	SAVE EP PSW;	S-A	A		
		=3180	SAVE EP R0;	A, 0100	VON		
		=3181	SAVE EP TOP-OF-STACK IN EP PC;	A, 0100	VON		
		=3182	RETURN.	CALL	JR		
0500	744F	=3183	STSAVE: CALL EPERK	A, 0100	VON		
0502	2303	=3184	MOV R, #3	CALL	JR		
0504	3400	=3185	CALL OUTUL	CALL	JR		
0506	7450	=3186	CALL OVSWP	CALL	JR		
0508	808F	=3187	MOV R0, #LOW(OV0BAS+OVSIZE)	CALL	JR		
050A	746A	=3188	CALL OVLORD	A, 0100	VON		
050C	8A20	=3189	ORL P2, #00100000B	ORL	P2, #00100000B		
050E	2314	=3190	MOV R, #14H	MOV	R, #14H		
0510	91	=3191	MOVX @R1, A	MOVX	@R1, A		
0511	9ADF	=3192	ANL P2, #NOT 00100000B	ANL	P2, #NOT 00100000B		
0513	8903	=3193	ORL P1, #00000011B	ORL	P1, #00000011B		
0515	F40B	=3194	CALL EPSTEP	CALL	JR		
0517	8A20	=3195	ORL P2, #00100000B	ORL	P2, #00100000B		
0519	9ACF	=3196	ANL P2, #NOT 00010000B	ANL	P2, #NOT 00010000B		
051B	8903	=3197	ORL P1, #KENBRAM OR ENBLNK	ORL	P1, #KENBRAM OR ENBLNK		
051D	F40B	=3198	CALL EPSTEP	CALL	JR		
		=3199					
		=3200	EXECUTION PROCESSOR IS NOW AT LOCATION 009H INTERNAL WITH				
		=3201	(RETURN ADDRESS+2) PUSHED ON STACK.				
		=3202					
051F	B0A5	=3203	MOV R0, #LOW(OV3BAS+OVSIZE)	MOV	R0, #LOW(OV3BAS+OVSIZE)		
0521	746A	=3204	CALL OVLORD	CALL	JR		
0523	F40B	=3205	CALL EPPASS	CALL	JR		
		=3206	MMOV EPACC, A	MMOV	EPACC, A		
0525	B920	=3219+	MOV R1, #EPACC	MOV	R1, #EPACC		
0527	A1	=3220+	MOV @R1, A	MOV	@R1, A		
0528	F40B	=3224	CALL EPPASS	CALL	JR		
		=3225	MMOV EPTIMR, A	MMOV	EPTIMR, A		
052A	B922	=3238+	MOV R1, #EPTIMR	MOV	R1, #EPTIMR		
052C	A1	=3239+	MOV @R1, A	MOV	@R1, A		
052D	F40B	=3243	CALL EPPASS	CALL	JR		
		=3244	MMOV EPPSW, A	MMOV	EPPSW, A		
052F	B921	=3257+	MOV R1, #EPPSW	MOV	R1, #EPPSW		
0531	A1	=3258+	MOV @R1, A	MOV	@R1, A		
0532	F40B	=3262	CALL EPPASS	CALL	JR		
		=3263	MMOV EPR0, A	MMOV	EPR0, A		
0534	B923	=3276+	MOV R1, #EPR0	MOV	R1, #EPR0		
0536	A1	=3277+	MOV @R1, A	MOV	@R1, A		
0537	D8BB	=3281	MOV R0, #LOW(OV1BAS+OVSIZE)	MOV	R0, #LOW(OV1BAS+OVSIZE)		
0539	746A	=3282	CALL OVLORD	CALL	JR		
		=3283	MMOV A, EPPSW	MMOV	A, EPPSW		
053B	B921	=3292+	MOV R1, #EPPSW	MOV	R1, #EPPSW		
053D	F1	=3293+	MOV A, @R1	MOV	A, @R1		
053E	07	=3297	DEC A	DEC	A		
053F	5307	=3298	ANL A, #07H	ANL	A, #07H		

LOC	OBJ	LINE	SOURCE STATEMENT	THIRTYTWO	THIRTYTWO	THIRTYTWO
0541	E7	=3299	RL A			
0542	0300	=3300	ADD A, #08H			
		=3301	MNOV SMALO, A			
0544	B930	=3314+	MOV RL, #SMALO			
0546	A1	=3315+	MOV @RL, A			
0547	F4B7	=3319	CALL EPFET			
0549	03FE	=3320	ADD R, #2			
054B	AA	=3321	MOV LDATA, A			
		=3322	MNOV EPPCLO, A			
054C	B924	=3335+	MOV RL, #EPPCLO			
054E	A1	=3336+	MOV @RL, A			
054F	F4C3	=3340	CALL EPSTOR			
0551	B930	=3341	MOV RL, #SMALO			
0553	11	=3342	INC @RL			
0554	F4B7	=3343	CALL EPFET			
0556	AA	=3344	MOV LDATA, A			
0557	53F0	=3345	ANL A, #11110000B			
0559	2A	=3346	XCH A, LDATA			
055A	13FF	=3347	ADDC A, #-1			
055C	530F	=3348	ANL A, #00001111B			
		=3349	MNOV EPPCHI, A			
055E	B925	=3362+	MOV RL, #EPPCHI			
0560	A1	=3363+	MOV @RL, A			
0561	4A	=3367	ORL A, LDATA			
0562	AA	=3368	MOV LDATA, A			
0563	F4C3	=3369	CALL EPSTOR			
0565	B825	=3370	MOV R0, #EPPCHI			
0567	347C	=3371	CALL UPDAD1			
0569	2340	=3372	MOV A, #01000000B ; "-" FOR DISPLAY			
056B	D4D8	=3373	CALL WDISP			
056D	B820	=3374	MOV R0, #EPACC			
056F	3490	=3375	CALL DSPMID			
0571	03	=3376	RET			
		=3377	SIZECHK			
0072		=3380+ SIZE SET 114				
		=3381+;				
		=3382+; *****				
		=3391 \$EJECT				

LOC	OBJ	LINE	SOURCE STATEMENT	THIRDTIME 33002	LOC	OBJ
		3392 \$	INCLUDE(=F0:HFILE.MOD)			
0000		=3393	CHARCR EQU 0DH ;CR			
000A		=3394	CHARLF EQU 0AH ;LF			
001A		=3395	CNTRLZ EQU 1AH ;CONTROL-2			
		=3396	;			
		=3397	CODEBLK 00			
0297		=3412+	ORG 663			
		=3416	HRECIN HEXFILE RECORD INPUT ROUTINE			
0297 34CD		=3417	HRECIN: CALL CHARIN			
0299 D31A		=3418	XRL A,#CNTRLZ			
029B C6E0		=3419	JZ DONE			
029D D31A		=3420	XRL A,#CNTRLZ			
029F D33A		=3421	XRL A,#('')			
02A1 9697		=3422	JNZ HRECIN			
		=3423	MMOV CHKSUM,ZERO			
02A3 D000		=3428+	MOV CHKSUM,#ZERO			
02A5 14F0		=3432	CALL BYTEIN			
		=3433	MMOV BUFcnt,A			
02A7 B941		=3446+	MOV RL,#BUFcnt			
02A9 A1		=3447+	MOV @RL,A			
02AA 14F0		=3451	CALL BYTEIN			
		=3452	MMOV SMAHI,A			
02AC B931		=3465+	MOV RL,#SMAHI			
02AE A1		=3466+	MOV @RL,A			
02AF 14F0		=3470	CALL BYTEIN			
		=3471	MMOV SMALO,A			
02B1 B930		=3484+	MOV RL,#SMALO			
02B3 A1		=3485+	MOV @RL,A			
02B4 14F0		=3489	CALL BYTEIN			
		=3490	MMOV RECTYP,A			
02B6 B942		=3503+	MOV RL,#RECTYP			
02B8 A1		=3504+	MOV @RL,A			
		=3508	;			
		=3509	HDATAIN HEX DATA BYTE IN			
		=3510	HDATAIN: MMOV A,BUFcnt			
02B9 B941		=3519+	MOV RL,#BUFcnt			
02BB F1		=3520+	MOV A,@RL			
02BC C6CC		=3524	JZ RECDON			
02BE 14F0		=3525	CALL BYTEIN			
02C0 AA		=3526	MOV LDATA,A			
02C1 F400		=3527	CALL LSTORE			
02C3 34F2		=3528	CALL INCSMA			
		=3529	MDEC BUFcnt			
02C5 B941		=3534+	MOV RL,#BUFcnt			
02C7 F1		=3535+	MOV A,@RL			
02C8 07		=3539+	DEC A			
02C9 A1		=3544+	MOV @RL,A			
02CA 44B9		=3547	JMP HDATAIN			
		=3548	;			
02CC 34CD		=3549	RECDON: CALL CHARIN			
02CE D33F		=3550	XRL A,#('')			
02D0 C6DB		=3551	JZ CKSMOK			
02D2 D33F		=3552	XRL A,#('')			
02D4 34BA		=3553	CALL NIBIN2			
02D6 14F2		=3554	CALL BYTE11			


```

=3555 ; (RESULT FOR NON-?? CHARACTERS IS AS IF
=3556 ; BYTEIN WAS CALLED.)
=3557 MMOV A,CHKSUM
02D8 FD =3573+ MOV A,CHKSUM
02D9 9GE1 =3577 JNZ CHKERR
=3578 CKSMOK: MMOV A,RECTYP
02DB B942 =3587+ MOV R1,RECTYP
02DD F1 =3588+ MOV A,R1
02DE C697 =3592 JZ HRECIN
=3593 ;
=3594 ; DONE HEX FILE CORRECTLY RECEIVED
02E0 83 =3595 DONE: RET
=3596 ;
=3597 ; CHKERR CHECKSUM ERROR IN INPUT RECORD DETECTED
02E1 B80C =3598 CHKERR: MOV LDAT0,#0CH
02E3 249A =3599 JMP PLRORR
=3600 SIZECHK
004E =3603+ SIZE SET 78
=3604+;
=3605+; *****
=3614 ;
=3615 CODEBLK 12
00F0 =3620+ ORG 240
=3624 ; BYTEIN BYTE INPUT SUBROUTINE.
=3625 ; RECEIVES TWO HEXIDECIMAL CHARACTERS FROM THE TAPE INPUT DEVICE
=3626 ; AND ASSEMBLES THEM INTO A SINGLE BYTE OF DATA:
00F0 34B8 =3627 BYTEIN: CALL NIBIN
00F2 47 =3628 BYTE1: SWAP A
00F3 AA =3629 MOV LDAT0,A
00F4 34B8 =3630 CALL NIBIN
=3631 MORL LDAT0,A
00F6 4A =3640+ ORL A,LDAT0
00F7 AA =3650+ MOV LDAT0,A
00F8 6D =3664 ADD A,CHKSUM
00F9 AD =3665 MOV CHKSUM,A
00FA FA =3666 MOV A,LDAT0
00FB 83 =3667 RET
=3668 SIZECHK
00FC =3671+ SIZE SET 12
=3672+;
=3673+; *****
=3682 ;
=3683 CODEBLK 25
01B8 =3693+ ORG 440
=3697 ; NIBIN RECEIVES A HEXIDECIMAL CHARACTER AND PRODUCES A MASKED FOUR BIT VALUE.
=3698 ; NOTE- ERROR CHECKING DONE TO VERIFY HEXIDECIMAL VALIDITY
01B8 34CD =3699 NIBIN: CALL CHARIN
01BA 03C6 =3700 NIBIN2: ADD A,#-3AH ; ACC=0F6-0FF FOR CHARACTERS '0'-'9'
=3701 ; ; CHARACTERS > '9' PRODUCE OVENFLOW
01BC E6C2 =3702 JNC NIBI3
01BE 03F9 =3703 ADD A,#-7 ; ACC=0-5 FOR CHARACTERS 'A'-'F'
01C0 E6C3 =3704 JNC ASCERR ; ERROR IF CHARACTER BETWEEN '9' AND 'A'
=3705 ;
=3706 ; ACC=0F6H-05H FOR CHARACTERS '0'-'F'
=3707 ;

```

LOC	OBJ	LINE	SOURCE STATEMENT	THIRDTAT2 33RU02	129 301
01C2	03FA	=3708	NIBI3: ADD A, #6 ; ACC=0F0H-0FFH FOR CHARACTERS '0'-'F'		
01C4	0310	=3709	ADD A, #10H ; ACC=00H-0FH FOR CHARACTERS '0'-'F';		
		=3710	OVENFLOW IF ABOVE IS TRUE		
01C6	E6C9	=3711	JNC ASCERR		
01C8	83	=3712	RET		
		=3713 ;			
		=3714 ;	ASCERR ILLEGAL HEXIDECIMAL CHARACTER RECEIVED		
01C9	8A0A	=3715	ASCERR: MOV LDATA, #0AH		
01CD	249A	=3716	JMP PERROR		
		=3717	SIZECHK		
0015		=3720H	SIZE SET 21		
		=3721+;			
		=3722+;	*****		
		=3731 ;			
		=3732 ;			
		=3733	CODEBLK 5		
01CD		=3743+	ORG 461		
		=3747 ;	CHARIN CHARACTER INPUT ROUTINE		
		=3748 ;	RECEIVES ONE ASCII CHARACTER FROM THE LOGICAL READER DEVICE.		
01CD	D449	=3749	CHARIN: CALL CIN		
01CF	537F	=3750	ANL A, #7FH		
01D1	83	=3751	RET		
		=3752	SIZECHK		
0005		=3755+	SIZE SET 5		
		=3756+;			
		=3757+;	*****		
		=3766 ;			
		=3767 ;			
		=3768	\$EJECT		

LOC	OBJ	LINE	SOURCE STATEMENT	THIRTYTWO EIGHT	LOC	OBJ
		=3769	CODEBLK 100			
0572		=3794	ORG 1394			
		=3798	HFILCO HEX FILE OUTPUT SUBROUTINE			
		=3799	WHEN CALLED WITH F0=0 OUTPUT IS STANDARD HEX FILE FORMAT.			
		=3800	WHEN CALLED WITH F0=1 OUTPUT IS FORMATTED DATA DUMP TO CRT.			
		=3801	HFILCO: MMOV MEMHI, SMARI			
0572 B931		=3817+	MOV R1, #SMARI			
0574 F1		=3818+	MOV A, @R1			
0575 B935		=3824+	MOV R1, #MEMHI			
0577 A1		=3825+	MOV @R1, A			
		=3826	MMOV MEMLO, SMALO			
0578 B930		=3844+	MOV R1, #SMALO			
057A F1		=3845+	MOV A, @R1			
057B B934		=3851+	MOV R1, #MEMLO			
057D A1		=3852+	MOV @R1, A			
		=3855	MMOV CHKSUM, ZERO			
057E B000		=3860+	MOV CHKSUM, #ZERO			
0580 B065		=3864	MOV R0, #HEXBUF			
		=3865				
		=3866	LDBYTE LOAD NEXT BYTE FROM MEMORY INTO HEX BUFFER			
0582 14FC		=3867	LDBYTE: CALL LFETCH			
0584 FA		=3868	MOV A, LDATA			
0585 A0		=3869	MOV @R0, A			
0586 18		=3870	INC R0			
0587 B4E2		=3871	CALL CHPMAS			
0589 E696		=3872	JNC ENDFIL			
058B 34F2		=3873	CALL INCSMA			
058D F8		=3874	MOV A, R0			
058E 0388		=3875	ADD A, #-(BUFLN+HEXBUF)			
0590 E682		=3876	JNC LDBYTE			
0592 D400		=3877	CALL HRECO			
0594 A472		=3878	JMP HFILCO			
		=3879				
		=3880	ENDFIL END HEX FILE TRANSMISSION:			
		=3881	PRINT OUT BUFFER FOR LAST DATA RECORD			
		=3882	PRINT OUT CANNED 'END-OF-FILE' RECORD			
		=3883	RETURN			
0596 D400		=3884	ENDFIL: CALL HRECO			
0598 B6A7		=3885	JF0 HFDONE			
059A 34D2		=3886	CALL TCRLFO			
059C B8AE		=3887	MOV R0, #-(LOW EOFREC)			
059E F8		=3888	ENDF1: MOV A, R0			
059F A3		=3889	MOV A, @R			
05A0 C0A7		=3890	JZ HFDONE			
05A2 B4BD		=3891	CALL CHARO			
05A4 18		=3892	INC R0			
05A5 A49E		=3893	JMP ENDF1			
05A7 34D2		=3894	HFDONE: CALL TCRLFO			
05A9 231A		=3895	MOV A, #CNTRLZ			
05AB B4BD		=3896	CALL CHARO			
05AD 83		=3897	RET			
		=3898				
		=3899	EOFREC CHARACTER SKING FOR CANNED END-OF-FILE RECORD FOR			
		=3900	INTEL HEX FILE FORMAT STANDARD.			
05AE 203A308		=3901	EOFREC: DB '0000001FF'			

LOC	OBJ	LINE	SOURCE STATEMENT	INSTRUCTIONS	LOC
0582	30303030				
0586	30314646				
058A	00	=3902	DB 0 ;END OF STRING CODE BYTE		
		=3903	SIZECHK		
0049		=3906+	SIZE SET 73		
		=3907+			
		=3908+	*****		
		=3917 ;			
		=3918 ;			
		=3919	CODEBLK 90		
0600		=3949+	ORG 1536		
		=3953 ;	HRECO HEXIDECEMAL RECORD OUTPUT SEQUENCE.		
		=3954 ;	HEX BUFFER ALREADY LOADED.		
0600 F8		=3955 HRECO:	MOV A, R0		
0601 0398		=3956	ADD A, #HEXBUF		
		=3957	MNOV BUFCONT, A		
0603 B941		=3970+	MOV RL, #BUFCONT		
0605 A1		=3971+	MOV @R1, A		
0606 34D2		=3975	CALL TCRLEO		
0608 2320		=3976	MOV A, #'		
060A 048D		=3977	CALL CHARO		
060C 0617		=3978	JF0 FDUMP1		
060E 233A		=3979	MOV A, #'		
0610 048D		=3980	CALL CHARO		
		=3981	MNOV A, BUFCONT		
0612 B941		=3990+	MOV RL, #BUFCONT		
0614 F1		=3991+	MOV A, @R1		
0615 34DE		=3995	CALL BYTEO		
		=3996 FDUMP1:	MNOV A, MEMHI		
0617 B935		=4005+	MOV RL, #MEMHI		
0619 F1		=4006+	MOV A, @R1		
061A 34DB		=4010	CALL BYTEO		
		=4011	MNOV A, MEMLO		
061C B934		=4020+	MOV RL, #MEMLO		
061E F1		=4021+	MOV A, @R1		
061F 34DB		=4025	CALL BYTEO		
0621 B620		=4026	JF0 FDUMP2		
0623 27		=4027	CLR A		
0624 34DB		=4028	CALL BYTEO		
0626 C42C		=4029	JMP DATO		
0628 233D		=4030 FDUMP2:	MOV A, #'		
062A 048D		=4031	CALL CHARO		
		=4032 ;	DATO DATA OUTPUT		
062C B865		=4033 DATO:	MOV R0, #HEXBUF		
062E B632		=4034 DATO1:	JF0 FDUMP5		
0630 C436		=4035	JMP FDUMP3		
0632 2320		=4036 FDUMP5:	MOV A, #'		
0634 048D		=4037	CALL CHARO		
0636 F0		=4038 FDUMP3:	MOV A, @R0		
0637 34DB		=4039	CALL BYTEO		
0639 18		=4040	INC R0		
		=4041	MDJNZ BUFCONT, DATO1		
063A B941		=4046+	MOV RL, #BUFCONT		
063C F1		=4047+	MOV A, @R1		
063D 07		=4051+	DEC A		

LOC	OBJ	LINE	SOURCE STATEMENT	STATEMENT	LOC	OBJ
063E	R1	=4056+	MOV R1, A		0000	0000
063F	962E	=4060+	JNZ DAT01		0000	0000
		=4062 ;			0000	0000
		=4063 ;	ENDREC END RECORD BEING TRANSMITTED		0000	0000
0641	B648	=4064	ENDREC: JF0 FDUMP4		0000	0000
		=4065	MMOV A, CHKSUM		0000	0000
0643	FD	=4081+	MOV A, CHKSUM		0000	0000
0644	37	=4085	CPL A		0000	0000
0645	17	=4086	INC A		0000	0000
0646	34DB	=4087	CALL BYTE0		0000	0000
0648	83	=4088	FDUMP4: RET		0000	0000
		=4089	SIZECHK		0000	0000
0049		=4092+	SIZE SET 73		0000	0000
		=4093+;			0000	0000
		=4094+;	*****		0000	0000
		=4103 ;			0000	0000
		=4104	CODEBLK 9		0000	0000
01D2		=4114+	ORG 466		0000	0000
		=4118 ;	TCRLF0 TAPE <CR><LF> OUTPUT		0000	0000
01D2	230D	=4119	TCRLF0: MOV A, #CHARCR		0000	0000
01D4	B4BD	=4120	CALL CHAR0		0000	0000
01D6	230H	=4121	MOV A, #CHARLF		0000	0000
01D8	B4BD	=4122	CALL CHAR0		0000	0000
01DA	83	=4123	RET		0000	0000
		=4124	SIZECHK		0000	0000
0089		=4127+	SIZE SET 9		0000	0000
		=4128+;			0000	0000
		=4129+;	*****		0000	0000
		=4138 ;			0000	0000
		=4139	CODEBLK 11		0000	0000
01DB		=4149+	ORG 475		0000	0000
		=4153 ;	BYTE0 BYTE OUTPUT		0000	0000
01DB	FA	=4154	BYTE0: MOV A, LDAT0		0000	0000
01DC	6D	=4155	ADD A, CHKSUM		0000	0000
01DD	AD	=4156	MOV CHKSUM, A		0000	0000
01DE	FA	=4157	MOV A, LDAT0		0000	0000
01DF	47	=4158	SWAP A		0000	0000
01E0	B4DB	=4159	CALL NIB0		0000	0000
01E2	FA	=4160	MOV A, LDAT0		0000	0000
01E3	B4DB	=4161	CALL NIB0		0000	0000
01E5	83	=4162	RET		0000	0000
		=4163	SIZECHK		0000	0000
008B		=4166+	SIZE SET 11		0000	0000
		=4167+;			0000	0000
		=4168+;	*****		0000	0000
		=4177 ;			0000	0000
		=4178	CODEBLK 12		0000	0000
01E6		=4188+	ORG 486		0000	0000
		=4192 ;	HEXASC HEXIDECIMAL NIBBLE TO ASCII CHARACTER CONVERSION.		0000	0000
01E6	530F	=4193	HEXASC: ANL A, #0FH		0000	0000
01E8	03F6	=4194	ADD A, #(-10)		0000	0000
01EA	F6EF	=4195	JC HEXNIB		0000	0000
01EC	033A	=4196	ADD A, #(-10+'0')		0000	0000
01EE	83	=4197	RET		0000	0000
01EF	0341	=4198	HEXNIB: ADD A, #('A')		0000	0000

LOC	OBJ	LINE	SOURCE STATEMENT	ASSEMBLY STATEMENT	LOC	OBJ
01F1	83	=4199	RET	CODEBLK 40	0000	
		=4200	SIZECHK	ORG 1257		
000C		=4203+	SIZE SET 12	ORG 1257		
		=4204+		ORG 1257		
		=4205+	*****	ORG 1257		
		=4214 ;		ORG 1257		
		=4215 ;		ORG 1257		
		=4216	DECLARE BIT50,CONST	ORG 1257		
000B		=4230	BIT50 EQU 11 ; DATA BITS PUT OUT (INCLUDING TWO STOP BITS)	ORG 1257		
		=4231 ;		ORG 1257		
		=4232	CODEBLK 30	ORG 1257		
04C9		=4252+	ORG 1225	ORG 1257		
		=4256 ;	HBDLAY HALF-BIT TIME DELAY	ORG 1257		
		=4257	HBDLAY: MOV H,HBITHI	ORG 1257		
04C9 B927		=4273+	MOV R1,HBITHI	ORG 1257		
04CB F1		=4274+	MOV R1,0R1	ORG 1257		
04CC B945		=4280+	MOV R1,0H	ORG 1257		
04CE R1		=4281+	MOV 0R1,A	ORG 1257		
		=4284	MOV R1,HBITHI	ORG 1257		
04CF B926		=4300+	MOV R1,HBITHI	ORG 1257		
04D1 F1		=4301+	MOV R1,0R1	ORG 1257		
04D2 A9		=4314+	MOV R1,A	ORG 1257		
04D3 B4D7		=4317	JMP 0R1	ORG 1257		
04D5 B900		=4318	HBD2: MOV R1,0R1	ORG 1257		
04D7 E9D7		=4319	HBD1: DJNZ R1,HBD1	ORG 1257		
		=4320	DJNZ H,HBD2	ORG 1257		
04D9 B945		=4325+	MOV R1,0H	ORG 1257		
04DB F1		=4326+	MOV R1,0R1	ORG 1257		
04DC 07		=4330+	DEC A	ORG 1257		
04DD R1		=4335+	MOV R1,0R1	ORG 1257		
04DE 96D5		=4339+	JNZ HBD2	ORG 1257		
04E0 83		=4341	RET	ORG 1257		
		=4342	SIZECHK	ORG 1257		
0018		=4345+	SIZE SET 24	ORG 1257		
		=4346+		ORG 1257		
		=4347+	*****	ORG 1257		
		=4356 ;		ORG 1257		
		=4357	\$EJECT	ORG 1257		

LOC	OBJ	LINE	SOURCE STATEMENT	DISASSEMBLY	ADDRESS
		=4358	CODEBLK 40		
0588		=4383+	ORG 1467		
		=4387 ; NIBO	MASK ACC TO MAKE HEX NIBBLE, TRANSLATE TO ASCII AND OUTPUT		
0588 34E6		=4388 NIBO:	CALL HEXASC		
		=4389 ;			
		=4390 ; CHARD	CONSOLE OUTPUT SUBROUTINE		
		=4391 ;	WRITES THE CONTENTS OF THE ACC TO THE CRT DISPLAY SCREEN		
		=4392 CHARD:	MOV REGC, A		
058D B944		=4405+	MOV R1, #REGC		
058F A1		=4406+	MOV R1, A		
		=4410	MOV B, BIT50 ; SET NUMBER OF BITS TO BE TRANSMITTED		
05C0 B943		=4421+	MOV R1, #B		
05C2 B108		=4422+	MOV R1, #BIT50		
05C4 97		=4426	CLR C ; CLEAR CARRY		
05C5 F6CB		=4427 C01:	JC C02		
05C7 99BF		=4428	RHL P1, #NOT TTYOUT		
05C9 A4CF		=4429	JMP C03		
05CB B940		=4430 C02:	ORL P1, #TTYOUT		
05CD 00		=4431	NOP ; EVEN OUT TWO BRANCH EXECUTION TIMES		
05CE 00		=4432	NOP		
05CF 94C9		=4433 C03:	CALL HBDLAY		
05D1 94C9		=4434	CALL HBDLAY		
05D3 97		=4435	CLR C ; SET WHAT WILL EVENTUALLY BECOME A STOP BIT		
05D4 A7		=4436	CPL C		
		=4437	MRRC REGC ; ROTATE CHARACTER RIGHT ONE BIT,		
05D5 B944		=4442+	MOV R1, #REGC		
05D7 F1		=4443+	MOV A, R1		
05D8 67		=4447+	RRC A		
05D9 A1		=4452+	MOV R1, A		
		=4455	; MOVING NEXT DATA BIT INTO CARRY		
		=4456	MOJNZ B, C01 ; CHECK IF CHARACTER (AND STOP BIT(S)) DONE		
05DA B943		=4461+	MOV R1, #B		
05DC F1		=4462+	MOV A, R1		
05DD 07		=4466+	DEC A		
05DE A1		=4471+	MOV R1, A		
05DF 96C5		=4475+	JNZ C01		
05E1 83		=4477	RET		
		=4478	SIZECHK		
0027		=4481+ SIZE	SET 39		
		=4482+;			
		=4483+;	*****		
		=4492 ;			
		=4493	CODEBLK 47		
0649		=4523+	ORG 1609		
		=4527 ; CIN	CONSOL INPUT SUBROUTINE WAITS FOR A KEYSTROKE (AND		
		=4528 ;	RETURNS WITH 8 BITS IN REG ACC.		
0649 B943		=4529 CIN:	MOV R1, #B		
064B B108		=4530	MOV R1, #8 ; DATA BITS TO BE READ		
064D 464D		=4531 C10:	JNT1 C10		
064F 464D		=4532	JNT1 C10		
0651 5651		=4533 C11:	JT1 C11		
0653 5651		=4534	JT1 C11		
0655 94C9		=4535	CALL HBDLAY		
0657 5651		=4536	JT1 C11		
0659 94C9		=4537 C12:	CALL HBDLAY		

LOC. OBJ	LINE	SOURCE STATEMENT	THIRTYTWO EIGHT	LOC. OBJ
065B 94C9	=4538	CALL HBDLAY	CODELK 25	
065D 5662	=4539	JT1 C13 ;CHECK SID LINE LEVEL	CODELK 25	
065F 97	=4540	CLR C ;DATA BIT IN CY	CODELK 25	
0660 C465	=4541	JMP C14	CODELK 25	
0662 97	=4542 C13:	CLR C	CODELK 25	
0663 A7	=4543	CPL C	CODELK 25	
0664 00	=4544	NOP ;EVEN OUT BRANCH EXECUTION TIMES	CODELK 25	
0665 00	=4545 C14:	NOP	CODELK 25	
0666 00	=4546	NOP	CODELK 25	
0667 00	=4547	NOP	CODELK 25	
	=4548	MRRC REGC	CODELK 25	
0668 B944	=4553+	MOV R1, #REGC	CODELK 25	
066A F1	=4554+	MOV A, @R1	CODELK 25	
066B 67	=4558+	RRC A	CODELK 25	
066C A1	=4563+	MOV @R1, A	CODELK 25	
	=4566	MOJNZ B, C12	CODELK 25	
066D B943	=4571+	MOV R1, #B	CODELK 25	
066F F1	=4572+	MOV A, @R1	CODELK 25	
0670 07	=4576+	DEC A	CODELK 25	
0671 A1	=4581+	MOV @R1, A	CODELK 25	
0672 9659	=4585+	JNZ C12	CODELK 25	
	=4587	MNOV A, REGC	CODELK 25	
0674 B944	=4596+	MOV R1, #REGC	CODELK 25	
0676 F1	=4597+	MOV A, @R1	CODELK 25	
0677 83	=4601	RET ; CHARACTER COMPLETE	CODELK 25	
062F	=4605+	SIZE SET 47	CODELK 25	
	=4606+	SIZECHK	CODELK 25	
	=4607+	*****	CODELK 25	
	=4616	\$EJECT	CODELK 25	

LOC	OBJ	LINE	SOURCE STATEMENT	SYMBOL TABLE	DISPATCH	LOC	OBJ
		4617 \$	INCLUDE(:F0:MEMREF.MOD)	VAL00H	100		
		=4618	CODEBLK 15	100	100		
02E5		=4633+	ORG 741	100	100		
		=4637 ;	CONFIL COMMAND TO FILL ADDRESS SPACE BETWEEN SMA AND EMA WITH DATA				
		=4638 ;	IN LOW BYTE OF MEM.				
		=4639 CONFIL:	MNOV LDATA, MEMLO				
02E5 B934		=4655+	MOV RL, MEMLO				
02E7 F1		=4656+	MOV A, RL				
02E8 AA		=4669+	MOV LDATA, A				
02E9 F400		=4672 LFILL:	CALL LSTORE				
02EB B4E2		=4673	CALL CNPMS				
02ED E6F3		=4674	JNC LFILL1				
02EF 34F2		=4675	CALL INCSMA				
02F1 44E9		=4676	JMP LFILL				
02F3 83		=4677 LFILL1:	RET				
		=4678	SIZECHK				
000F		=4681+ SIZE	SET 15				
		=4682+;					
		=4683+;	*****				
		=4692 ;					
		=4693	CODEBLK 4				
00FC		=4690+	ORG 52				
		=4702 ;	LFETCH FETCHES CONTENTS OF LOGICAL MEMORY ADDRESS DETERMINED BY				
		=4703 ;	<TYPE>, <SMAHI>, & <SMALO> INTO <LDATA>.				
00FC D478		=4704 LFETCH:	CALL AFETCH				
00FE AA		=4705	MOV LDATA, A				
00FF 83		=4706	RET				
		=4707	SIZECHK				
0004		=4710+ SIZE	SET 4				
		=4711+;					
		=4712+;	*****				
		=4721 ;					
		=4722	CODEBLK 75				
0678		=4752+	ORG 1656				
		=4756 ;					
		=4757 ;	AFETCH LOGICAL FETCH SUBROUTINE				
		=4758 ;	FETCHS CONTENTS OF VARIOUS MEMORY SPACES TO ACC.				
		=4759 AFETCH:	MNOV A, TYPE				
0678 B937		=4768+	MOV RL, #TYPE				
067A F1		=4769+	MOV A, RL				
067B 037E		=4773	ADD A, #LOW LFETBL				
067D B3		=4774	JMPP 0A				
		=4775 ;					
067E 04		=4776 LFETBL:	DB LOW LFEPH				
067F 98		=4777	DB LOW LFEDM				
0680 9C		=4778	DB LOW LFEREG				
0681 A9		=4779	DB LOW LFEINT				
0682 B1		=4780	DB LOW LFEBRK				
0683 B1		=4781	DB LOW LFEBRK				
		=4782 ;					
		=4783 LFEPH:	MNOV A, SMAHI				
0684 B931		=4792+	MOV RL, #SMAHI				
0686 F1		=4793+	MOV A, RL				
0687 9698		=4797	JNZ LFEDM				
		=4798	MNOV A, SMALO				

LOC	OBJ	LINE	SOURCE STATEMENT	INSTRUCTION	OPERANDS	DISPLACEMENT	ADDRESS
0689	B930	=4887+	MOV R1, #SMALO	MOV	R1, #SMALO		00000000
068B	F1	=4888+	MOV R1, #SMALO	MOV	R1, #SMALO		00000000
068C	03E9	=4812	ADD A, #OVSZ	ADD	A, #OVSZ		00000000
068E	F698	=4813	JC LFEDM	JC	LFEDM		00000000
		=4814	MNOV R1, #SMALO	MNOV	R1, #SMALO		00000000
0690	B930	=4823+	MOV R1, #SMALO	MOV	R1, #SMALO		00000000
0692	F1	=4824+	MOV R1, #SMALO	MOV	R1, #SMALO		00000000
0693	03AE	=4820	ADD A, #OVSZ	ADD	A, #OVSZ		00000000
0695	A9	=4829	MOV R1, A	MOV	R1, A		00000000
0696	F1	=4830	MOV R1, #SMALO	MOV	R1, #SMALO		00000000
0697	03	=4831	RET	RET			00000000
0698	94E1	=4832 LFEDM:	CALL LPSEL	CALL	LPSEL		00000000
069A	01	=4833	MOVX A, R1	MOVX	A, R1		00000000
069B	03	=4834	KET	KET			00000000
		=4835 ;					00000000
		=4836 LFEREC:	MNOV R1, #SMALO	MNOV	R1, #SMALO		00000000
069C	B930	=4845+	MOV R1, #SMALO	MOV	R1, #SMALO		00000000
069E	F1	=4846+	MOV R1, #SMALO	MOV	R1, #SMALO		00000000
069F	537F	=4850	ANL A, #01111111B ; CHECK IF LOW 7 BITS = 0	ANL	A, #01111111B		00000000
06A1	C6A5	=4851	JZ LFER0	JZ	LFER0		00000000
06A3	E4B7	=4852	JMP EPFET	JMP	EPFET		00000000
		=4853 ;					00000000
		=4854 LFER0:	MNOV R1, #EPR0	MNOV	R1, #EPR0		00000000
06A5	B923	=4863+	MOV R1, #EPR0	MOV	R1, #EPR0		00000000
06A7	F1	=4864+	MOV R1, #EPR0	MOV	R1, #EPR0		00000000
06A8	03	=4868	RET	RET			00000000
		=4869 ;					00000000
		=4870 LFEINT:	MNOV R1, #SMALO	MNOV	R1, #SMALO		00000000
06A9	B930	=4879+	MOV R1, #SMALO	MOV	R1, #SMALO		00000000
06AB	F1	=4880+	MOV R1, #SMALO	MOV	R1, #SMALO		00000000
06AC	0320	=4884	ADD A, #EPACC	ADD	A, #EPACC		00000000
06AE	A9	=4885	MOV R1, A	MOV	R1, A		00000000
06AF	F1	=4886	MOV R1, #SMALO	MOV	R1, #SMALO		00000000
06B0	03	=4887	RET	RET			00000000
		=4888 ;					00000000
		=4889 ; LFEBRK LOGICAL FETCH OF BREAK-POINT DATA					00000000
06B1	94E1	=4890 LFEBRK:	CALL LPSEL	CALL	LPSEL		00000000
06B3	99F7	=4891	ANL P1, #NOT 00001000B	ANL	P1, #NOT 00001000B		00000000
06B5	8908	=4892	ORL P1, #00001000B	ORL	P1, #00001000B		00000000
06B7	99FD	=4893	ANL P1, #NOT 00000010B	ANL	P1, #NOT 00000010B		00000000
06B9	8901	=4894	ORL P1, #00000001B	ORL	P1, #00000001B		00000000
06BB	01	=4895	MOVX A, R1	MOVX	A, R1		00000000
06BC	2301	=4896	MOV R1, #01H	MOV	R1, #01H		00000000
06BE	86C1	=4897	JNI LFEBR1	JNI	LFEBR1		00000000
06C0	27	=4898	CLR A	CLR	A		00000000
06C1	03	=4899 LFEBR1:	KET	KET			00000000
		=4900	SIZECHK	SIZECHK			00000000
004A		=4903+ SIZE SET 74					00000000
		=4904+;					00000000
		=4905+; *****					00000000
		=4914 \$EJECT					00000000

LOC	OBJ	LINE	SOURCE STATEMENT	THINKSTAT2 33002	WJ	LOC	OBJ
		=4915	CODEBLK 85	00000000	WON	0000	0000
0700		=4950+	ORG 1792	1000	WON	0000	0000
		=4954 ;		00000000	WON	0000	0000
		=4955 ;	LSTORE LOGICAL STORE SUBROUTINE	00000000	WON	0000	0000
		=4956 ;	STORES CONTENTS OF LDATA INTO VARIOUS MEMORY SPACES.	00000000	WON	0000	0000
		=4957 LSTORE: MMOV	A, TYPE	00000000	WON	0000	0000
0700 B937		=4966+	MOV RL, #TYPE	00000000	WON	0000	0000
0702 F1		=4967+	MOV A, @RL	00000000	WON	0000	0000
0703 0306		=4971	ADD A, #LOW LSTBL	00000000	WON	0000	0000
0705 83		=4972	JMPP @A	00000000	WON	0000	0000
		=4973 ;		00000000	WON	0000	0000
0706 0C		=4974 LSTBL: DB	LOW LSTPM	00000000	WON	0000	0000
0707 21		=4975	DB LOW LSTDM	00000000	WON	0000	0000
0708 26		=4976	DB LOW LSTREG	00000000	WON	0000	0000
0709 34		=4977	DB LOW LSTINI	00000000	WON	0000	0000
070H 3D		=4978	DB LOW LSTBRK	00000000	WON	0000	0000
070C 3D		=4979	DB LOW LSTBRK	00000000	WON	0000	0000
		=4980 ;		00000000	WON	0000	0000
		=4981 LSTPM: MMOV	A, #SMALI	00000000	WON	0000	0000
070C B931		=4990+	MOV RL, #SMALI	00000000	WON	0000	0000
070E F1		=4991+	MOV A, @RL	00000000	WON	0000	0000
070F 9621		=4995	JNZ LSTDM	00000000	WON	0000	0000
		=4996	MMOV A, #SMALO	00000000	WON	0000	0000
0711 B938		=5005+	MOV RL, #SMALO	00000000	WON	0000	0000
0713 F1		=5006+	MOV A, @RL	00000000	WON	0000	0000
0714 03E9		=5010	ADD A, #OVSZC	00000000	WON	0000	0000
0716 F621		=5011	JC LSTDM	00000000	WON	0000	0000
		=5012	MMOV A, #SMALO	00000000	WON	0000	0000
0718 B938		=5021+	MOV RL, #SMALO	00000000	WON	0000	0000
071A F1		=5022+	MOV A, @RL	00000000	WON	0000	0000
071B 034E		=5026	ADD A, #OVBIF	00000000	WON	0000	0000
071D A9		=5027	MOV RL, A	00000000	WON	0000	0000
071E FA		=5028	MOV A, LDATA	00000000	WON	0000	0000
071F A1		=5029	MOV @RL, A	00000000	WON	0000	0000
0720 83		=5030	RET	00000000	WON	0000	0000
		=5031 ;		00000000	WON	0000	0000
0721 94E1		=5032 LSTDM: CALL	LPGSEL	00000000	WON	0000	0000
0723 FA		=5033	MOV A, LDATA	00000000	WON	0000	0000
0724 91		=5034	MOVX @RL, A	00000000	WON	0000	0000
0725 83		=5035	RET	00000000	WON	0000	0000
		=5036 ;		00000000	WON	0000	0000
		=5037 LSTREG: MMOV	A, #SMALO	00000000	WON	0000	0000
0726 B938		=5046+	MOV RL, #SMALO	00000000	WON	0000	0000
0728 F1		=5047+	MOV A, @RL	00000000	WON	0000	0000
0729 537F		=5051	ANL A, #01111111B ; CHECK IF LOW ORDER BITS = 0	00000000	WON	0000	0000
072B C62F		=5052	JZ LSTR0	00000000	WON	0000	0000
072D E4C3		=5053	JMP EPSTOR	00000000	WON	0000	0000
		=5054 ;		00000000	WON	0000	0000
		=5055 LSTR0: MMOV	EPSTOR, LDATA	00000000	WON	0000	0000
072F FA		=5078+	MOV A, LDATA	00000000	WON	0000	0000
0730 B923		=5084+	MOV RL, #EPR0	00000000	WON	0000	0000
0732 A1		=5085+	MOV @RL, A	00000000	WON	0000	0000
0733 83		=5088	RET	00000000	WON	0000	0000
		=5089 ;		00000000	WON	0000	0000
		=5090 LSTINT: MMOV	A, #SMALO	00000000	WON	0000	0000

LOC	OBJ	LINE	SOURCE STATEMENT	SYMBOL TABLE	DISPATCH	DISPATCH
0734	B930	=5099+	MOV R1, #SMALO	CODEBLK 11	0000	0000
0736	F1	=5100+	MOV A, R1	0000	0000	0000
0737	0320	=5104	ADD A, #EPACC	0000	0000	0000
0739	A9	=5105	MOV R1, A	0000	0000	0000
073A	FA	=5106	MOV A, LDATA	0000	0000	0000
073B	A1	=5107	MOV R1, A	0000	0000	0000
073C	83	=5108	RET	0000	0000	0000
		=5109 ;				
		=5110 ;	LSTBRK LOGICAL STORE OF BREAK-POINT DATA			
073D	94E1	=5111	LSTBRK: CALL LPGSEL	0000	0000	0000
073F	FA	=5112	MOV A, LDATA	0000	0000	0000
0740	1246	=5113	JB0 LSTBR1	0000	0000	0000
0742	8901	=5114	ORL P1, #00000001B	0000	0000	0000
0744	E448	=5115	JMP LSTBR2	0000	0000	0000
0746	99FE	=5116	LSTBR1: ANL P1, #NOT 00000001B	0000	0000	0000
0748	99F7	=5117	LSTBR2: ANL P1, #NOT 00001000B	0000	0000	0000
074A	81	=5118	MOVX A, R1	0000	0000	0000
074B	8908	=5119	ORL P1, #00001000B	0000	0000	0000
074D	83	=5120	RET	0000	0000	0000
		=5121	SIZECHK			
004E		=5124+ SIZE SET 78				
		=5125+;				
		=5126+; *****				
		=5135 ;				
		=5136	CODEBLK 17			
04E1		=5156+ ORG 1249				
		=5160 ;	LPGSEL LOGICAL PAGE SELECT.			
		=5161 ;	SETS UP PORT 2 TO ADDRESS APPROPRIATE BYTE OF RAM BLOCK.			
		=5162	LPGSEL: MMOV A, TYPE			
04E1	B937	=5171+	MOV R1, #TYPE			
04E3	F1	=5172+	MOV A, R1			
04E4	5301	=5176	ANL A, #00000001B			
04E6	47	=5177	SWAP A			
		=5178	MORL A, #SMHI			
04E7	B931	=5184+	MOV R1, #SMHI			
04E9	41	=5185+	ORL A, R1			
04EH	4340	=5189	ORL A, #01000000B			
04EC	3A	=5190	OUTL P2, A			
		=5191	MMOV A, #SMALO			
04ED	B930	=5200+	MOV R1, #SMALO			
04EF	F1	=5201+	MOV A, R1			
04F0	A9	=5205	MOV R1, A			
04F1	83	=5206	RET			
		=5207	SIZECHK			
0011		=5210+ SIZE SET 17				
		=5211+;				
		=5212+; *****				
		=5221 ;				
		=5222	#EJECT			

LOC	OBJ	LINE	SOURCE STATEMENT	THEIR STATE	330002	LOC	OBJ
		=5223	CODEBLK 11				
01F2		=5233+	ORG 498				
		=5237	INCSMA INCREMENT STARTING MEMORY ADDRESS WORD.				
01F2 B930		=5238	INCSMA: MOV R1, #SMALO				
01F4 11		=5239	INCH: INC @R1				
01F5 F1		=5240	MOV A, @R1				
01F6 96FC		=5241	JNZ INCH1				
01F8 19		=5242	INC R1				
01F9 F1		=5243	MOV A, @R1				
01FA 17		=5244	INC A				
01FB 31		=5245	XCHD A, @R1				
01FC 83		=5246	INCH1: RET				
		=5247	SIZECHK				
0008		=5250+	SIZE SET 11				
		=5251+					
		=5252+	*****				
		=5261					
		=5262	CODEBLK 12				
02F4		=5277+	ORG 756				
		=5281	DECSMA DECREMENT SMA WORD.				
02F4 B930		=5282	DECSMA: MOV R1, #SMALO				
02F6 F1		=5283	MOV A, @R1				
02F7 07		=5284	DEC A				
02F8 21		=5285	XCH A, @R1				
02F9 96FF		=5286	JNZ DECSM1				
02FB 19		=5287	INC R1				
02FC F1		=5288	MOV A, @R1				
02FD 07		=5289	DEC A				
02FE 31		=5290	XCHD A, @R1				
02FF 83		=5291	DECSM1: RET				
		=5292	SIZECHK				
000C		=5295+	SIZE SET 12				
		=5296+					
		=5297+	*****				
		=5306					
		=5307	CODEBLK 15				
05E2		=5332+	ORG 1506				
		=5336	CHPMAS COMPARE MEMORY ADDRESSES				
		=5337	COMPARE SMA BYTES WITH EMA BYTES TO DETERMINE RELATIVE MAGNITUDE.				
		=5338	RETURNS WITH CARRY=1 IFF <SMA> >= <EMA>				
		=5339	IS CALLED AFTER ACTION HAS BEEN PERFORMED ON <SMA> TO DETERMINE IF				
		=5340	TASK IS COMPLETED:				
		=5341	IF CY=0 THEN <SMA> >= <EMA> ==> TERMINATE TASK.				
		=5342	IF CY=1 THEN <SMA> < <EMA> ==> INC SMA AND REPEAT.				
		=5343	CHPMAS: MMOV A, SMALO				
05E2 B930		=5352+	MOV R1, #SMALO				
05E4 F1		=5353+	MOV A, @R1				
05E5 37		=5357	CPL A				
		=5358	MADD A, EMALO				
05E6 B932		=5364+	MOV R1, #EMALO				
05E8 61		=5365+	ADD A, @R1				
		=5369	MMOV A, SMAHI				
05E9 B931		=5378+	MOV R1, #SMAHI				
05EB F1		=5379+	MOV A, @R1				
05EC 37		=5383	CPL A				

LOC	OBJ	LINE	SOURCE STATEMENT	TARGET2	COND2	VAL2	LOC	OBJ
		5411 \$	INCLUDE(:F0:KBD.MOD)	INAME.A	COND	VAL		
		=5412	CODEBLK 100	INAME.A	VAL	VAL	0000	
074E		=5447+	ORG 1870	INAME.A	COND	VAL	0000	
		=5451 ;						
		=5452 ;	KEYBOARD AND DISPLAY PROCESSING ROUTINE					
		=5453 ;	CALLED PERIODICALLY WHEN KBD AND DISPLAY ARE TO BE ALIVE.					
074E D5		=5454 TIINT:	SEL R01					
		=5455	MMOV ASAVE, A					
074F B93E		=5468+	MOV R1, #ASAVE					
0751 A1		=5469+	MOV @R1, A					
0752 23F0		=5473	MOV A, #(-10H)					
0754 62		=5474	MOV T, A ;RELOAD TIMER INTERVAL					
0755 27		=5475	CLR A					
0756 3E		=5476	MOVD PSEGH1, A ;WRITE BLANK PATTERN TO SEG DRIVERS					
0757 3D		=5477	MOVD PSEGLO, A					
0758 FD		=5478	MOV A, CURDIG					
0759 07		=5479	DEC A					
075A 3F		=5480	MOVD PDIGIT, A ;ENERGIZE CHARACTER					
075B 0C		=5481	MOVD A, PINPUT ;LOAD ANY SWITCH CLOSURES					
075C AA		=5482	MOV ROTPAT, A					
		=5483	;WRITE NEXT SEGMENT PATTERN					
075D FD		=5484	MOV A, CURDIG					
075E 07		=5485	DEC A					
075F 0346		=5486	ADD A, #SEGMAP ;ADD CURDIG DISPLACEMENT TO BASE					
0761 A8		=5487	MOV R0, A					
0762 F0		=5488	MOV A, @R0 ;LOAD ACC W/ NEXT SEGMENT PATTERN					
0763 3D		=5489	MOVD PSEGLO, A ;ENABLE APPROPRIATE SEGMENTS					
0764 47		=5490	SWAP A					
0765 3E		=5491	MOVD PSEGH1, A					
		=5492 ;						
		=5493 ;	*****					
		=5494 ;	THE NEXT CHARACTER IS NOW BEING DISPLAYED.					
		=5495 ;	THE KEYBOARD SCAN ROUTINE IS INTEGRATED INTO THE DISPLAY SCAN.					
		=5496 ;	WITH THE CURRENT ROW ENERGIZED, CHECK IF THERE ARE ANY INPUTS.					
		=5497 ;	*****					
		=5498 ;						
		=5499 ;	ROTATE BITS THROUGH THE CY WHILE INCREMENTING KEYLOC.					
		=5500 ;						
0766 B804		=5501	MOV ROTCNT, #NCOLS ;SET UP FOR (NCOLS) LOOPS THROUGH 'NXTLOC'					
		=5502 NXTLOC:	MRRC ROTPAT					
0768 FA		=5514+	MOV A, ROTPAT					
0769 67		=5518+	RRC A					
076A AA		=5529+	MOV ROTPAT, A					
076B F68B		=5532	JC SCANS ;ONE BIT IN CY INDICATES KEY NOT DOWN					
076D BE01		=5533	MOV KEYFLG, #1 ;MARK THAT AT LEAST ONE KEY WAS DETECTED					
		=5534	; \ IN THE CURRENT SCAN					
		=5535 ;						
		=5536 ;	*****					
		=5537 ;	A KEYSTROKE WAS DETECTED FOR THE CURRENT COLUMN. ITS					
		=5538 ;	POSITION IS IN REGISTER KEYLOC. SEE IF SAME KEY SENSED LAST CYCLE.					
		=5539 ;	*****					
		=5540 ;						
		=5541	MMOV A, KEYLOC					
076F B93C		=5550+	MOV R1, #KEYLOC					
0771 F1		=5551+	MOV A, @R1					

LOC	OBJ	LINE	SOURCE STATEMENT	TARGET STATEMENT	LOC
0772	2C	=5555	XCH A, LASTKY		
0773	DC	=5556	XRL A, LASTKY		
0774	C67C	=5557	JZ SCAN3		
		=5558 ;			
		=5559 ;	*****		
		=5560 ;	A DIFFERENT KEY WAS READ ON THIS CYCLE THAN ON THE PREVIOUS CYCLE.		
		=5561 ;	SET NREPTS TO THE DEBOUNCE PARAMETER FOR A NEW COUNTDOWN.		
		=5562 ;	*****		
		=5563 ;			
0776	B93D	=5564	MOV R1, #NREPTS		
0778	B106	=5565	MOV @R1, #6		
077A	E48B	=5566	JMP SCAN5		
		=5567 ;			
		=5568 ;	*****		
		=5569 ;	SAME KEY WAS DETECTED 35 ON PREVIOUS CYCLE		
		=5570 ;	LOOK AT NREPTS: IF ALREADY ZERO, DO NOTHING.		
		=5571 ;	ELSE DECREMENT NREPTS.		
		=5572 ;	IF THIS RESULTS IN ZERO, MOVE LASTKY INTO KBDUF.		
		=5573 ;	*****		
		=5574 ;			
		=5575	SCAN3: MMOV A, NREPTS		
077C	B93D	=5584+	MOV R1, #NREPTS		
077E	F1	=5585+	MOV A, @R1		
077F	C68B	=5589	JZ SCAN5 ; IF ALREADY ZERO		
0781	07	=5590	DEC A ; INDICATE ONE MORE SUCCESSIVE KEY DETECTION		
		=5591	MMOV NREPTS, A		
0782	B93D	=5604+	MOV R1, #NREPTS		
0784	A1	=5605+	MOV @R1, A		
0785	968B	=5609	JNZ SCAN5 ; IF DECREMENT DOES NOT RESULT IN ZERO		
		=5610	MMOV KBDUF, LASTKY ; TO MARK NEW KEY CLOSURE		
0787	FC	=5633+	MOV A, LASTKY		
0788	B93B	=5639+	MOV R1, #KBDUF		
078A	A1	=5640+	MOV @R1, A		
		=5643 ;			
078B	B93C	=5644	SCAN5: MOV R1, #KEYLOC		
078D	11	=5645	INC @R1		
078E	ED68	=5646	DJNZ ROTCNT, NXTLOC		
0790	ED88	=5647	DJNZ CURDIG, TIRET1		
0792	B088	=5648	MOV CURDIG, #CHARNO		
		=5649 ;			
		=5650 ;	*****		
		=5651 ;	THE FOLLOWING CODE SEGMENT IS USED BY THE KEYBOARD SCANNING ROUTINE.		
		=5652 ;	IT IS EXECUTED ONLY AFTER A REFRESH SEQUENCE IS COMPLETED		
		=5653 ;	*****		
		=5654 ;			
		=5655	MMOV KEYLOC, ZERO		
0794	B93C	=5666+	MOV R1, #KEYLOC		
0796	B100	=5667+	MOV @R1, #ZERO		
0798	FE	=5671	MOV A, KEYFLG		
0799	969D	=5672	JNZ SCAN8 ; JUMP IF ANY KEYS WERE DETECTED		
		=5673	MMOV LASTKY, #NEG1 ; CHANGE <LASTKY> WHEN NO KEYS ARE DOWN		
079B	BCFF	=5678+	MOV LASTKY, #NEG1		
079D	BE00	=5682	SCAN8: MOV KEYFLG, #0		
		=5683 ;			
		=5684 ;	*****		

LOC	OBJ	LINE	SOURCE STATEMENT	THIRDS	THIRDS	THIRDS
		=5685 ;				
		=5686 ;	KBD/DISP RETURN CODE- RESTORES SYSTEM STATUS.			
		=5687	MMOV A, RDELAY			
079F	B93F	=5696+	MOV R1, #RDELAY			
07A1	F1	=5697+	MOV R, @R1			
07A2	CGA8	=5701	JZ TIRET1			
07A4	07	=5702	DEC A			
		=5703	MMOV KDELAY, A			
07A5	B93F	=5716+	MOV R1, #RDELAY			
07A7	A1	=5717+	MOV @R1, A			
		=5721 TIRET1:	MMOV A, ASAVE			
07A8	B93E	=5730+	MOV R1, #ASAVE			
07AA	F1	=5731+	MOV R, @R1			
07AB	93	=5735	RETR			
		=5736 ;				
		=5737 ;				
		=5738 ;	TOFPOL TIMER OVERFLOW POLLING SUBROUTINE			
		=5739 ;	CALLED REPEATEDLY FROM WHEREVER KBD/DISP MUST BE ALIVE			
		=5740 ;	MONITORS THE TIMER OVERFLOW FLAG (TOF) AND CALLS SERVICE			
		=5741 ;	ROUTINE WHEN APPROPRIATE.			
07AC	164E	=5742 TOFPOL:	JTF TIINT			
07AE	83	=5743	RET			
		=5744	SIZECHK			
0061		=5747+ SIZE SET 97				
		=5748+;				
		=5749+; *****				
		=5758 \$EJECT				

LOC	OBJ	LINE	SOURCE STATEMENT	THIRD STATEMENT	LOC	OBJ
		=5759	CODEBLK 17			
06C2		=5789+	ORG 1730			
		=5793 ;				
		=5794	KBDIN KEYBOARD INPUT SUBROUTINE.			
		=5795 ;	RETURNS ONLY AFTER A NEW KEYSTROKE HAS BEEN DETECTED AND DEBOUNCED.			
		=5796 ;	VALUE OF KEY POSITION IN SWITCH MATRIX IS			
		=5797 ;	RETURNED IN THE ACCUMULATOR.			
		=5798 ;	DISPLAY CHARACTER NOW ON BLANKED BEFORE RETURNING.			
06C2 DF03		=5799 KBDIN	MOV XPCODE, #3			
06C4 74D1		=5800	CALL XPTST			
06C6 F4AC		=5801 KBD11	CALL TOPPOL			
		=5802	MMOV A, KECBUF			
06C8 B93B		=5811	MOV R1, #KBDUF			
06CA F1		=5812	MOV A, @R1			
06CB F2C6		=5816	JB7 KBD11			
06CD 27		=5817	CLR A			
06CE 3E		=5818	MOVD PSEGL1, A			
06CF 3D		=5819	MOVD PSEGL0, A			
06D0 37		=5820	CPL A			
06D1 21		=5821	XCH A, @R1			
06D2 83		=5822	RET			
		=5823	SIZECHK			
0011		=5826+ SIZE	SET 17			
		=5827+;	*****			
		=5837 ;				
		=5838	CODEBLK 15			
05F1		=5863+	ORG 010 1521			
		=5867	CLEAR WRITES 'BLANK' CHARACTERS INTO ALL DISPLAY REGISTERS.			
		=5868 ;	RETURNS WITH NEXTPL SET TO LEFTMOST CHARACTER POSITION			
		=5869 ;	DOES NOT AFFECT ACC OR CY			
05F1 B846		=5870 CLEAR	MOV R0, #SEGMAP1			
05F3 B908		=5871	MOV R1, #CHARNO			
05F5 B000		=5872 DBLANK	MOV @R0, #0			
05F7 18		=5873	INC @R0			
05F8 E9F5		=5874	DJNZ @R1, DBLANK			
		=5875	MMOV @R1, NEXTPL			
05FA D93A		=5886+	MOV @R1, #NEXTPL			
05FC C108		=5887+	MOV @R1, #CHARNO			
05FE 83		=5891	RET			
		=5892	SIZECHK			
000E		=5895+ SIZE	SET 14			
		=5896+;	*****			
		=5906 ;				
		=5907	CODECLK 44			
06D3		=5937+	ORG 1747			
		=5941	DSFACC DISPLAY VALUE OF LOW NIBBLE OF ACC			
06D3 530F		=5942 DSFACC	ANL A, #0FH			
06D5 03EF		=5943	ADD A, #06PATS			
06D7 A3		=5944	MOV @R1, A			
		=5945	WDISP WRITES BIT PATTERN NOW IN ACC INTO NEXT CHARACTER POSITION			
		=5946 ;	OF THE DISPLAY (NEXTPL). INCREMENTS NEXTPL			
		=5947 ;	RESULTS IN DISPLAY BEING FILLED LEFT TO RIGHT, THEN RESTARTING			
06D8 AE		=5948 WDISP	MOV DSPTMP, A			

LOC	OBJ	LINE	SOURCE STATEMENT	COMPILED STATEMENT	LOC	OBJ
06D9	B04	=5949	MOV XPCODE, #4	CODEBLK 12		
06DB	74D1	=5950	CALL XPTST	CALL 1230		
		=5951	MOV A, NEXTPL			
06DD	B93A	=5960+	MOV R1, #NEXTPL			
06DF	F1	=5961+	MOV A, @R1			
06E0	0345	=5965	ADD A, #SEGMAP-1			
06E2	A9	=5966	MOV R1, A			
06E3	FE	=5967	MOV A, DSPTMP			
06E4	A1	=5968	MOV @R1, A			
		=5969	MOJNZ NEXTPL, WDISP1			
06E5	B93A	=5974+	MOV R1, #NEXTPL			
06E7	F1	=5975+	MOV A, @R1			
06E8	07	=5979+	DEC A			
06E9	A1	=5984+	MOV @R1, A			
06EA	96EE	=5988+	JNZ WDISP1			
06EC	B108	=5990	MOV @R1, #CHARNO			
06EE	83	=5991	WDISP1: RET			
		=5992 ;				
		=5993 ;	DGPATS IS THE BASE FOR THE TABLE OF SEGMENT PATTERNS FOR HEX DIGITS.			
		=5994 ;	HERE THE FULL HEX SET (0-F) IS INCLUDED.			
		=5995 ;				
00EF		=5996	DGPATS EQU \$ AND 0FFH			
		=5997 ;				
		=5998 ;	FORMAT IS PGFEDCBA IN STANDARD SEVEN-SEGMENT ENCODING CONVENTION			
		=5999 ;	WHERE P REPRESENTS THE DECIMAL POINT			
06EF	3F	=6000	DB 00111111B ; SEGMENT PATTERN FOR DIGIT '0'			
06F0	06	=6001	DB 00000110B ; SEGMENT PATTERN FOR DIGIT '1'			
06F1	58	=6002	DB 01011011B ; SEGMENT PATTERN FOR DIGIT '2'			
06F2	4F	=6003	DB 01001111B ; SEGMENT PATTERN FOR DIGIT '3'			
06F3	66	=6004	DB 01100110B ; SEGMENT PATTERN FOR DIGIT '4'			
06F4	6D	=6005	DB 01101101B ; SEGMENT PATTERN FOR DIGIT '5'			
06F5	7D	=6006	DB 01111101B ; SEGMENT PATTERN FOR DIGIT '6'			
06F6	07	=6007	DB 00000111B ; SEGMENT PATTERN FOR DIGIT '7'			
06F7	7F	=6008	DB 01111111B ; SEGMENT PATTERN FOR DIGIT '8'			
06F8	67	=6009	DB 01100111B ; SEGMENT PATTERN FOR DIGIT '9'			
06F9	77	=6010	DB 01110111B ; SEGMENT PATTERN FOR DIGIT 'A'			
06FA	7C	=6011	DB 01111100B ; SEGMENT PATTERN FOR DIGIT 'B'			
06FB	39	=6012	DB 00111001B ; SEGMENT PATTERN FOR DIGIT 'C'			
06FC	5E	=6013	DB 01011110B ; SEGMENT PATTERN FOR DIGIT 'D'			
06FD	79	=6014	DB 01111001B ; SEGMENT PATTERN FOR DIGIT 'E'			
06FE	71	=6015	DB 01110001B ; SEGMENT PATTERN FOR DIGIT 'F'			
		=6016	SIZECHK			
002C		=6019+	SIZE SET 44			
		=6020+				
		=6021+ ;	*****			
		=6030 ;				
		=6031	CODEBLK 12			
04F2		=6051+	ORG 1266			
		=6055 ;	DELAY SUBROUTINE WAITS FOR THE NUMBER OF COMPLETE			
		=6056 ;	DISPLAY SCANS CORRESPONDING TO THE ACC CONTENTS.			
		=6057 ;	USED WITH CRUDE HUMAN INTERFACES- AS WHEN OPERATOR SHOULD SEE			
		=6058 ;	SOME DISPLAY CHANGE WHILE IT IS CHANGING.			
		=6059	DELAY: MOV RDELAY, A			
04F2	B93F	=6072+	MOV R1, #RDELAY			
04F4	A1	=6073+	MOV @R1, A			

LOC	OBJ	LINE	SOURCE STATEMENT	OBJECT STATEMENT	LOC	OBJ
04F5	F4AC	=6077	DELAY1: CALL 10FFPOL	CALL 10FFPOL	04F5	F4AC
		=6078	MMOV A, RDELAY	MMOV A, RDELAY		
04F7	D93F	=6087+	MOV R1, RDELAY	MOV R1, RDELAY	04F7	D93F
04F9	F1	=6088+	MOV A, R1	MOV A, R1	04F9	F1
04FA	96F5	=6092	JNZ DELAY1	JNZ DELAY1	04FA	96F5
04FC	83	=6093	RET	RET	04FC	83
		=6094	SIZECHK	SIZECHK		
0008		=6097+	SIZE SET 11	SIZE SET 11	0008	
		=6098+				
		=6099+	*****	*****		
		=6100				
		=6109	CODEBLK 8	CODEBLK 8		
07AF		=6144+	ORG 1967	ORG 1967	07AF	
		=6148	KBDPOL POLL STATUS OF KEYBOARD INPUT ROUTINE.	KBDPOL POLL STATUS OF KEYBOARD INPUT ROUTINE.		
		=6149	RETURN WITH ACC BIT 7 = 0 IF KEYBOARD INPUT HAS BEEN RECEIVED.	RETURN WITH ACC BIT 7 = 0 IF KEYBOARD INPUT HAS BEEN RECEIVED.		
07AF	BF05	=6150	KBDPOL: MOV XPCODE, #5	KBDPOL: MOV XPCODE, #5	07AF	BF05
07B1	74D1	=6151	CALL XPTST	CALL XPTST	07B1	74D1
		=6152	MMOV A, KBDDBUF	MMOV A, KBDDBUF		
07B3	B938	=6161+	MOV R1, #KBDDBUF	MOV R1, #KBDDBUF	07B3	B938
07B5	F1	=6162+	MOV A, R1	MOV A, R1	07B5	F1
07B6	83	=6166	RET	RET	07B6	83
		=6167	SIZECHK	SIZECHK		
0008		=6170+	SIZE SET 8	SIZE SET 8	0008	
		=6171+				
		=6172+	*****	*****		
		=6181	\$EJECT	\$EJECT		

LOC	OBJ	LINE	SOURCE STATEMENT	INFORMATIONAL STATEMENT	LOC	OBJ
		6182 \$	INCLUDE(.F0:LINK.MOD)			
		=6183	CODEBLK 15			
07B7		=6218+	ORG 1975			
		=6222	EPFET: FETCH DATA BYTE FROM EP INTERNAL RAM ADDRESSED BY SMAIL0.			
		=6223	EPFET: MMOV A, SMAIL0			
07B7 B930		=6232+	MOV R1, #SMAIL0			
07B9 F1		=6233+	MOV A, @R1			
07BA F4D0		=6237	CALL EPPASS			
07BC 2380		=6238	MOV A, #10000000B			
07BE F4D0		=6239	CALL EPPASS			
07C0 F4D0		=6240	CALL EPPASS			
07C2 83		=6241	RET			
		=6242	SIZECHK			
000C		=6245+	SIZE SET 12			
		=6246+;				
		=6247+;	*****			
		=6256 ;				
		=6257	CODEBLK 15			
07C3		=6292+	ORG 1987			
		=6296	EPSTOR: STORE DATA IN LDATA IN EP INTERNAL RAM AT <SMAIL0>			
07C3 FA		=6297	EPSTOR: MOV A, LDATA			
07C4 F4D0		=6298	CALL EPPASS			
		=6299	MMOV A, SMAIL0			
07C6 B930		=6308+	MOV R1, #SMAIL0			
07C8 F1		=6309+	MOV A, @R1			
07C9 537F		=6313	ANL A, #01111111D			
07CB F4D0		=6314	CALL EPPASS			
07CD F4D0		=6315	CALL EPPASS			
07CF 83		=6316	RET			
		=6317	SIZECHK			
000D		=6320+	SIZE SET 13			
		=6321+;				
		=6322+;	*****			
		=6331	\$EJECT			

LOC	OBJ	LINE	SOURCE STATEMENT	INSTR	DISP	LOC
		=6332 ;	THE FOLLOWING UTILITIES INVOLVE INTERCHANGES BETWEEN THE MP AND EP.			
		=6333 ;				
		=6334	CODEBLK 11			
07D0		=6369+	ORG 2000			
		=6373 ;	EPPASS PASSES A SINGLE PARAMETER BYTE TO THE EP THROUGH THE LINK.			
		=6374 ;	WRITE THE CONTENTS OF THE ACC TO THE LINK.			
		=6375 ;	RELEASE THE EP.			
		=6376 ;	READ THE LINK INTO THE ACC.			
		=6377 ;	RETURN.			
07D0 8A30		=6378 EPPASS:	ORL P2, #00110000B			
07D2 91		=6379	MOVX @R1, A			
07D3 99FE		=6380	ANL P1, #NOT ENBRAM			
07D5 8902		=6381	ORL P1, #ENBLNK			
07D7 F4D8		=6382	CALL EPSTEP			
07D9 81		=6383	MOVX A, @R1			
07DA 83		=6384	RET			
		=6385	SIZECHK			
0008		=6388+ SIZE	SET 11			
		=6389+;				
		=6390+;	*****			
		=6399 ;				
		=6400	CODEBLK 25			
07D8		=6435+ ORG	2011			
		=6439 ;	EPSTEP RELEASES EP TO RUN IN PRESENT MODE UNTIL AN ANTICIPATED			
		=6440 ;	HARDWARE BREAK OCCURS.			
		=6441 ;	(DUE TO SINGLE STEPPING, LINK (OPCODE FETCH, OR LINK DATA FETCH.)			
		=6442 ;	MUST OCCUR WITHIN A FINITE NUMBER OF CYCLES (<40 MP CYCLES)			
		=6443 ;	OR WATCHDOG TIMER WILL ASSUME A COMMUNICATIONS ERROR			
		=6444 ;	BETWEEN THE MP AND EP.			
07D8 F4F4		=6445 EPSTEP:	CALL EPREL			
07D0 B90A		=6446	MOV R1, #10			
07DF 86F1		=6447 EPSTE1:	JNI EPSTE2			
07E1 E9D1		=6448	DJNZ R1, EPSTE1			
07E3 8910		=6449	ORL P1, #EPRSET			
07E5 744F		=6450	CALL EPBRK			
07E7 B88B		=6451	MOV R0, #LOW(OVLBAS+OVSZ)			
07E9 746A		=6452	CALL OVLOAD			
07EB 99EF		=6453	ANL P1, #NOT EPRSET			
07ED B70E		=6454	MOV LDATA, #0EH			
07EF 249A		=6455	JMP PERROR			
07F1 744F		=6456 EPSTE2:	CALL EPBRK			
07F3 83		=6457	RET			
		=6458	SIZECHK			
0019		=6461+ SIZE	SET 25			
		=6462+;				
		=6472 ;				
		=6473 ;				
		=6474 \$EJECT				

LOC	OBJ	LINE	SOURCE STATEMENT	THIRTYTWO SOURCE	THIRTYTWO	LOC	OBJ
		=6475	CODEBLK 9	THE FOLLOWING UTILITIES INVOLVE INTERFACING TO THE			
07F4		=6510+	ORG 2036				
		=6514	EPREL RELEASES EP TO RUN IN PRESENT MODE.				
		=6515	SEQUENCE IS AS FOLLOWS:				
		=6516	PUT MEMORY ARRAY IN EP MODE;				
		=6517	RAISE /SSTEP;				
		=6518	RETURN.				
07F4 99F7		=6519	EPREL: ANL P1,#NOT CLRBF				
07F6 8908		=6520	ORL P1,#CLRBF				
07F8 9ABF		=6521	ANL P2,#NOT 01000000B				
07FA 8904		=6522	ORL P1,#00000100B				
07FC 83		=6523	RET				
		=6524	SIZECHK				
0009		=6527+	SIZE SET 9				
		=6528+					
		=6529+	*****				
		=6530					
		=6539					
		=6540	CODEBLK 11				
034F		=6500+	ORG 047				
		=6504	EPBRK REGAIN CONTROL OF MEMORY ARRAY FROM EP.				
		=6505	DROP /SSTEP;				
		=6506	WAIT 30 USECS.;				
		=6507	PUT MEMORY ARRAY IN MP MODE;				
		=6508	RETURN.				
034F 99FB		=6509	EPBRK: ANL P1,#NOT 00000100B				
0351 8920		=6510	ORL P1,#MODOUT				
0353 8905		=6511	MOV R1,#5				
0355 E955		=6512	DJNZ R1,\$				
0357 8A40		=6513	ORL P2,#01000000B				
0359 83		=6514	RET				
		=6515	SIZECHK				
0008		=6516+	SIZE SET 11				
		=6517+					
		=6600+	*****				
		=6609					
		=6610					
		=6611	CODEBLK 16				
035A		=6651+	ORG 858				
		=6655	OVSMP OVERLAY SWAP.				
		=6656	SWAPS BLOCK OF DATABYTES (USER'S PROGRAM) BETWEEN MP RAM & EP-PM.				
035A 0865		=6657	OVSMP: MOV R0,#OVBUF+OVSZ				
035C 0917		=6658	MOV R1,#OVSZ				
035E 2340		=6659	MOV R,#01000000B				
0360 3A		=6660	OUTL P2,A				
0361 C0		=6661	OVSML: DEC R0				
0362 C9		=6662	DEC R1				
0363 01		=6663	MOVX @R1				
0364 20		=6664	XCH R,R0				
0365 91		=6665	MOVX @R1,A				
0366 F9		=6666	MOV A,R1				
0367 9661		=6667	JNZ OVSML				
0369 83		=6668	RET				
		=6669	SIZECHK				
0010		=6672+	SIZE SET 16				

LOC	OBJ	LINE	SOURCE STATEMENT	SYMBOL	ADDRESS	HEX
		=6673+;				
		=6674+; *****				
		=6683 ;				
		=6684 ; CODEBLK 14				
036A		=6724+ ORG 074				
		=6728 ; OVLORD OVERLAY LOAD				
		=6729 ; MOVES BLOCK OF DATA BYTES (ASSEMBLED SOURCE) FROM PG3 TO EP PM				
		=6730 ; TOP OF DATA BLOCK LOADED AND BLOCK LENGTH DETERMINED BY R0 AND R1				
036A 0917		=6731 OVLORD: MOV R1, #OVSZ				
036C 2340		=6732 MOV R, #010000000				
036E 3A		=6733 OUTL P2, R				
036F C8		=6734 MML01: DEC R0				
0370 C9		=6735 DEC R1				
0371 F8		=6736 MOV R, R0				
0372 E3		=6737 MOV P3, R, R				
0373 91		=6738 MOVX R1, R				
0374 F9		=6739 MOV R, R1				
0375 966F		=6740 JNZ MML01				
0377 83		=6741 RET				
		=6742 SIZECHK				
000E		=6745+ SIZE SET 14				
		=6746+;				
		=6747+; *****				
		=6756 \$EJECT				

LOC	OBJ	LINE	SOURCE STATEMENT	THINKSTAT2 33H002	10: 30J
		=6757 ;			
		=6758 ;	=====		
		=6759 ;			
		=6760 ;	THE REST OF THIS MODULE CONTAINS THE MINI-MONITORS WHICH OVERLAY		
		=6761 ;	THE EMULATION PROCESSOR PROGRAM RAM TO GIVE THE		
		=6762 ;	MASTER PROCESSOR ACCESS TO INTERNAL REGISTERS AND RAM OF THE EP.		
		=6763 ;	=====		
		=6764 ;			
		=6765 ;			
		=6766	DATABLK 22		
0378		=6771+	ORG 808		
		=6775 ;			
		=6776 ;	OVERLAY TO BREAK EP EXECUTION AND JUMP TO LOCATION 009H.		
		=6777 ;	LOCATION 009H REACHED WITH TOP-OF-STACK = RETURN ADDRESS+2		
		=6778 ;	DUE TO FORCED "CALL" DURING WHICH PC WAS INCREMENTED.		
		=6779 ;	LOCS 003H & 007H CALL 009H TO SIMULATE SAME CONDITION		
		=6780 ;	IF BREAK OCCURS DURING INTERRUPT CYCLE.		
		=6781 ;	SOURCE CODE FOR MINI-MONITOR OVERLAYED OVER LOW ORDER PROGRAM RAM.		
		=6782 ;			
0378		=6783	OV0BAS EQU \$		
0378		=6784	ORG OV0BAS		
0378 1409		=6785	CALL 009H		
037A 00		=6786	NOP		
		=6787 ;			
037B		=6788	ORG OV0BAS+003H		
037B 1409		=6789	CALL 009H		
037D 00		=6790	NOP		
037E 00		=6791	NOP		
		=6792 ;			
037F		=6793	ORG OV0BAS+007H		
037F 1409		=6794	CALL 009H		
0381 00		=6795	NOP		
0382 00		=6796	NOP		
0383 00		=6797	NOP		
0384 00		=6798	NOP		
0385 00		=6799	NOP		
0386 00		=6800	NOP		
0387 00		=6801	NOP		
0388 00		=6802	NOP		
0389 00		=6803	NOP		
038A 00		=6804	NOP		
038B 00		=6805	NOP		
		=6806 ;			
038C		=6807	ORG OV0BAS+014H		
038C 0403		=6808	JMP 009H		
		=6809 ;			
		=6810	SIZECHK		
0016		=6813+	SIZE SET 22		
		=6814+;			
		=6815+;	*****		
		=6824	\$EJECT		

LOC	OBJ	LINE	SOURCE STATEMENT	THIRTYTWO SOURCE	LOC	OBJ
		=6825	DATABLK 22	SS JUMPTO		
038E		=6830+	ORG 910	SS JUMPTO		
		=6834 ;				
		=6835 ; OV3-	OVERLAY TO SAVE STATUS DATA AFTER BREAK			
		=6836 ;	ACC, TIMER/COUNTER, PSM (WITH F1), & RAM LOC 0 PASSED SEQUENTIALLY			
		=6837 ;	TO MP.			
		=6838 ;	SOURCE CODE FOR MINI-MONITOR OVERLAYED OVER LOW ORDER PROGRAM RAM			
		=6839 ;				
038E		=6840 OV3BAS	EQU \$			
038E		=6841 ORG	OV3BAS			
038E 0400		=6842	JMP 006H			
0390 00		=6843	NOP			
		=6844 ;				
0391		=6845 ORG	OV3BAS+003H			
0391 83		=6846	RET			
0392 00		=6847	NOP			
0393 00		=6848	NOP			
0394 00		=6849	NOP			
		=6850 ;				
0395		=6851 ORG	OV3BAS+007H			
0395 83		=6852	RET			
039C 00		=6853	NOP			
		=6854 ;				
0397		=6855 ORG	OV3BAS+009H			
0397 90		=6856	MOVX @R0, A			
0398 42		=6857	MOV A, T			
0399 90		=6858	MOVX @R0, A			
039A C7		=6859	MOV A, PSM			
039B 7611		=6860	JF1 OV3B1			
039D 53F7		=6861	ANL A, #11110111B			
0311		=6862 OV3B1	EQU \$- (LOW OV3BAS)			
039F 90		=6863	MOVX @R0, A			
03A0 C5		=6864	SEL RB0			
03A1 FC		=6865	MOV A, R0			
03A2 0409		=6866	JMP 009H			
		=6867 ;				
		=6868	SIZECHK			
0016		=6871+ SIZE	SET 22			
		=6872+;				
		=6873+; *****				
		=6882 \$EJECT				

LOC	OBJ	LINE	SOURCE STATEMENT	THIRTIETH SOURCE	LOC	OBJ
		=6944	DATABLK 23			
030A		=6949+	ORG 954			
		=6953 ;				
		=6954 ; OV2-	OVERLAY TO RESTORE EP STATUS SAVED ON BREAK AND RESUME USER'S PROGRAM.			
		=6955 ;	SOURCE CODE FOR MINI-MONITOR OVERLAYED OVER LOW ORDER PROGRAM RAM			
		=6956 ;				
030A		=6957 OV2BAS	EQU \$			
030A		=6958 ORG	OV2BAS			
030A 0400		=6959	JMP 000H			
030C 00		=6960	NOP			
		=6961 ;				
030D		=6962 ORG	OV2BAS+003H			
030D 03		=6963	RET			
030E 00		=6964	NOP			
030F 00		=6965	NOP			
0300 00		=6966	NOP			
		=6967 ;				
03C1		=6968 ORG	OV2BAS+007H			
03C1 03		=6969	RET			
03C2 00		=6970	NOP			
		=6971 ;				
03C3		=6972 ORG	OV2BAS+009H			
03C3 90		=6973	MOVX 0R0, A			
		=6974 ;				
03C4 00		=6975	MOVX A, 0R0			
03C5 00		=6976	MOV 0R0, A			
03C6 00		=6977	MOVX A, 0R0			
03C7 07		=6978	MOV PSW, A			
03C8 05		=6979	CLR F1			
03C9 05		=6980	CPL F1			
03CA 7213		=6981	JB3 OV2B1			
03CC 05		=6982	CLR F1			
		=6983 ;				
0313		=6984 OV2B1	EQU \$-LOW OV2BAS			
		=6985 ;				
03CD 00		=6986	MOVX A, 0R0			
03CE 62		=6987	MOV T, A			
03CF 00		=6988	MOVX A, 0R0			
03D0 93		=6989	RETR			
		=6990	SIZECHK			
0017		=6993+ SIZE SET 23				
		=6994+;				
		=6995+; *****				
		=7004 \$EJECT				

LOC	OBJ	LINE	SOURCE STATEMENT	TAG	STATUS	DATE	TIME
		7005 ;					
		7006	CODEBLK 11				
03D1		7046+	ORG 977				
03D1 0A80		7050	XPTEST: ORL P2, #80H				
03D3 0A		7051	IN A, P2				
03D4 9A7F		7052	ANL P2, # (NOT 80H)				
03D6 F2D9		7053	JB7 \$+3				
03D8 83		7054	RET				
03D9 F5		7055	SEL MB1				
03DA 0400		7056	JMP 800H				
		7057	SIZECHK				
000B		7060+	SIZE SET 11				
		7061+;					
		7062+;	*****				
		7071 ;					
		7072	CODEBLK 13				
03DC		7112+	ORG 988				
03DC 28432931		7116	DB '(C)1979 INTEL'				
03E0 39373920							
03E4 494E5445							
03E8 4C							
		7117	SIZECHK				
000D		7120+	SIZE SET 13				
		7121+;					
		7122+;	*****				
		7131 ;					
		7132 ;					
		7133	RSOURCE				
0100		7135+	PGSIZE SET ORGPG0-000H ; BYTES USED ON PAGE 0				
00FD		7136+	PGSIZE SET ORGPG1-100H ; BYTES USED ON PAGE 1				
0100		7137+	PGSIZE SET ORGPG2-200H ; BYTES USED ON PAGE 2				
00E9		7138+	PGSIZE SET ORGPG3-300H ; BYTES USED ON PAGE 3				
00FD		7139+	PGSIZE SET ORGPG4-400H ; BYTES USED ON PAGE 4				
00F7		7140+	PGSIZE SET ORGPG5-500H ; BYTES USED ON PAGE 5				
00FF		7141+	PGSIZE SET ORGPG6-600H ; BYTES USED ON PAGE 6				
00FD		7142+	PGSIZE SET ORGPG7-700H ; BYTES USED ON PAGE 7				
		7143+;EJECT					

LOC	OBJ	LINE	SOURCE STATEMENT	SYMBOL TABLE	LOC	OBJ
		7145 ;	*****			
		7146 ;				
		7147 ;	FILL ALL UNUSED MEMORY LOCATIONS WITH NOP OPCODES			
		7148 ;				
		7149 ;	*****			
		7150 ;				
		7151 \$GEN				
		7158 ;				
01FD		7160	ORG ORGPG1			
		7161	REPT (200H - ORGPG1)			
		7162	DB 0			
		7163	ENDM			
01FD 00		7164+	DB 0			
01FE 00		7165+	DB 0			
01FF 00		7166+	DB 0			
		7168 ;				
		7175 ;				
03E9		7177	ORG ORGPG3			
		7178	REPT (400H - ORGPG3)			
		7179	DB 0			
		7180	ENDM			
03E9 00		7181+	DB 0			
03EA 00		7182+	DB 0			
03EB 00		7183+	DB 0			
03EC 00		7184+	DB 0			
03ED 00		7185+	DB 0			
03EE 00		7186+	DB 0			
03EF 00		7187+	DB 0			
03F0 00		7188+	DB 0			
03F1 00		7189+	DB 0			
03F2 00		7190+	DB 0			
03F3 00		7191+	DB 0			
03F4 00		7192+	DB 0			
03F5 00		7193+	DB 0			
03F6 00		7194+	DB 0			
03F7 00		7195+	DB 0			
03F8 00		7196+	DB 0			
03F9 00		7197+	DB 0			
03FA 00		7198+	DB 0			
03FB 00		7199+	DB 0			
03FC 00		7200+	DB 0			
03FD 00		7201+	DB 0			
03FE 00		7202+	DB 0			
03FF 00		7203+	DB 0			
		7205 ;				
04FD		7207	ORG ORGPG4			
		7208	REPT (500H - ORGPG4)			
		7209	DB 0			
		7210	ENDM			
04FD 00		7211+	DB 0			
04FE 00		7212+	DB 0			
04FF 00		7213+	DB 0			
		7215 ;				
05FF		7217	ORG ORGPG5			
		7218	REPT (600H - ORGPG5)			

LOC	OBJ	LINE	SOURCE STATEMENT	LOC	OBJ
		7219	DB 0		
		7220	ENDM		
05FF	00	7221+	DB 0		
		7223 ;			
06FF		7225	ORG ORGPG6		
		7226	REPT (700H - ORGPG6)		
		7227	DB 0		
		7228	ENDM		
06FF	00	7229+	DB 0		
		7231 ;			
07FD		7233	ORG ORGPG7		
		7234	REPT (800H - ORGPG7)		
		7235	DB 0		
		7236	ENDM		
07FD	00	7237+	DB 0		
07FE	00	7238+	DB 0		
07FF	00	7239+	DB 0		
		7241 ;			
		7242 \$EJECT			

1-160

1-161

All trademarks copyrighted © Intel Corporation 1978.

1-165

All information copyrighted © Intel Corporation 1978.

PSGL0 000C	RDELAY 003F	RECDON 02CC	RECTYP 0042	KEGC 0044	REORG 0005	RERKOR 0190	RINT 0011
ROTCNT 0003	ROTPAT 0002	RSOURC 0012	SCAN3 077C	SCANS 078B	SCAN8 079D	SEGMAP 0046	SING 0010
SIZE 0000	SIZECH 0011	SNHHT 0031	SNALO 0030	STRCOM 001D	STGFC 002C	STRMEM 0026	STRTMP 0040
STRUL 0019	STWAVE 0500	TCLF0 01D2	TIINT 074E	TIKET1 07F8	TFPOL 079C	TIYOUT 0040	TYPE 0037
UPDAD1 017C	UPDADR 0178	VERNO 0029	WBRK 0016	WDISP 06DE	WDISP1 06EE	XPCODE 0007	XPIEST 03D1
ZERO 0000							
ASSEMBLY COMPLETE, NO ERRORS							

ISIS-II ASSEMBLER SYMBOL CROSS REFERENCE, V2.1																
NAME	DATA	DATA	DATA	DATA	DATA	DATA	DATA	DATA	DATA	DATA	DATA	DATA	DATA	DATA	DATA	DATA
20	165#	1614	1629	1637	1650	1658	1721	1787	1806	1818	1977	1985	1999	2177	2388	2444
	2546	2586	2719	2788	2796	2843	2857	2865	2893	2910	2928	2943	2958	2973	3007	3068
	3096	3207	3215	3226	3234	3245	3253	3264	3272	3288	3302	3310	3323	3331	3350	3358
	3434	3442	3453	3461	3472	3480	3491	3499	3515	3562	3583	3637	3958	3966	3986	4001
	4016	4070	4393	4401	4592	4764	4788	4803	4819	4841	4859	4875	4962	4986	5001	5017
	5042	5095	5167	5180	5196	5348	5360	5374	5386	5456	5464	5546	5580	5592	5600	5692
	5704	5712	5726	5807	5956	6060	6068	6083	6157	6228	6304					
?ASAVE	1235#	5460	5466	5722	5728											
?B	1280#	4413	4413	4413	4419	4459	4469	4569	4579							
?B0FNT	117#	746	754#	763	771#	780	788#	797	805#	814	822#	831	839#			
?B0R2	110#	754														
?B0R3	111#	771														
?B0R4	112#	788														
?B0R5	113#	805														
?B0R6	114#	822														
?B0R7	115#	839														
?B1PNT	126#	859	867#	880	888#	901	909#	922	930#	943	951#					
?B1R2	119#	867														
?B1R3	120#	888														
?B1R4	121#	909														
?B1R5	122#	930														
?B1R6	123#	951														
?B1R7	124#															
?BCODE	1163#	1561	1561	1561	1567	1610	1616	2390								
?BINOP	415#	1817	2387	2439	2909	3632	5179	5359	5385							
?BIT50	4217#	4411														
?BUFCN	1262#	3438	3444	3511	3517	3532	3542	3962	3968	3982	3988	4044	4054			
?BUFILE	649#															
?CHARN	585#	5876														
?CHKSU	791#	3426	3426	3426	3558	3564	3571	3571	3858	3858	3858	4066	4072	4079	4079	
?CONST	104#	585	586	590	594	601	602	606	610	617	618	622	626	633	634	638
	642	649	650	654	658	671	672	676	680	686	687	691	695	701	702	706
	710	716	717	721	725	4217	4218	4222	4226							
?CURDI	912#															
?DEBNC	617#															
?DSPTI	1037#	3092	3098													
?DSPTM	808#															
?EMPHI	1136#	5388														
?EMPHLO	1127#	5362														
?EPHCC	965#	2969	2975	3211	3217											
?EPHCH	1010#	2761	2777	2912	3354	3360										
?EPPCL	1001#	2734	2750	2806	2814	2819	3327	3333								
?EPPSW	974#	2792	2798	2839	2845	2894	2900	2939	2945	3249	3255	3284	3290			
?EPR0	992#	2924	2930	3268	3274	4655	4861	5060	5082							
?EPTIM	983#	2954	2960	3230	3236											
?FORM1	295#	1615	1634	1655	1688	1695	1722	1745	1780	1807	1819	1982	2000	2013	2178	2389
	2441	2445	2547	2587	2720	2735	2742	2762	2769	2793	2811	2818	2844	2862	2877	2899
	2911	2929	2944	2959	2974	3008	3069	3097	3212	3231	3250	3269	3289	3307	3320	3355
	3439	3458	3477	3496	3516	3531	3563	3504	3634	3638	3807	3814	3834	3841	3963	3987
	4002	4017	4043	4071	4263	4270	4290	4297	4322	4396	4439	4458	4550	4568	4593	4645
	4652	4765	4789	4804	4820	4842	4860	4876	4963	4987	5002	5018	5043	5061	5068	5096
	5168	5181	5197	5349	5361	5375	5387	5461	5504	5547	5581	5597	5616	5623	5693	5709
	5727	5808	5957	5971	6065	6084	6158	6229	6305							
?FORM2	319#	1638	1659	1692	1702	1986	2739	2749	2765	2776	2797	2815	2825	2866	3216	3235
	3254	3273	3311	3332	3359	3443	3462	3481	3500	3811	3821	3838	3848	3967	4267	4277
	4294	4304	4402	4649	4659	5065	5081	5465	5601	5620	5636	5713	6069			
?FORM3	339#	1755	2023	2452	2887	3541	3651	4053	4332	4449	4468	4560	4578	5520	5981	

© Intel Corporation 1978

LOC	OBJ	LINE	SOURCE STATEMENT
		7243	END
USER SYMBOLS			
2A	0004	2ASAVE	0002
2B	0002	2B	0002
2B0R6	0007	2B0PNT	0008
2B1R	0008	2B1PNT	0007
2CONST	0003	2B1R2	0003
2EPPCH	0002	2B1N0P	0022
2FORM4	001C	2B0DNC	0003
2HREGC	0002	2B0STI	0002
2KEYLO	0002	2CPR0	0002
2NCOLS	0003	2HBITH	0002
2PLU51	0003	2HREGD	0002
2REGC	0002	2HREGL	0002
2ROTCN	0001	2LDRAT	0000
2START	030C	2LNGT	0000
ASAVE	003E	2NEXTP	0002
BRKNKT	0234	2NREPT	0002
CGOPAT	0476	2R0T0P	0001
CHARO	058D	2RSAVE	0000
CIN	0649	2SEGNM	0003
CNTTBL	04A1	2TYPE	0002
COMGOR	0461	2UNARY	002A
DBLANK	05F5	2VBRK	0000
DECSML	02TF	2WERSN	0002
DGR	015D	2PCOD	0000
DPMEN	015F	2ZERO	0003
DSPH1	018E	2ZSIZ	000E
DWBK1	016D	2ZMAH1	0002
ENDFIL	0596	2ZINDX	0075
EPFET	07B7	2ZVBUF	0003
EPRESET	0010	2ZVSI2	0003
EPSSPT	0004	2ZRECT	0002
EXPM1	027B	2ZMALO	0002
FDUMP2	0628	2ZAFCH	0678
HBD2	04D5	2ZBRKIL	022E
HFONE	05A7	2ZCGNB	047C
HREGC	002E	2ZCHARN	0008
INPAD1	00C7	2ZC13	0662
JTGO	0220	2ZCMPT	05F0
KBDPOL	07AF	2ZCONTRL	001A
KEYGO	001E	2ZCOMFIL	02E5
KEYREG	001B	2ZDAT01	062E
LFEBR1	06C1	2ZDECLAR	0003
LFETCH	00FC	2ZDGPAT	00EF
LSTINT	0734	2ZDPA	0172
MADO	0024	2ZDSGNON	0137
MAINC1	0075	2ZDSS	01CF
MEMLO	0034	2ZENBRAM	0001
MRL	002E	2ZENDF1	059E
NEXTPL	003F	2ZEPCON1	041F
NUMCON	0038	2ZERROR2	0186
ORPG2	0300	2ZEXAM0	0250
OUTUTL	0100	2ZEXPON	0000
OV3BR5	038E	2ZFDUMP1	0617
PERROR	019A	2ZH	0045
		2ZHEXBUF	0665
		2ZHKEGC	002C
		2ZINIT	0000
		2ZJMTBL	0206
		2ZJT0FIL	0222
		2ZKBDI1	06C6
		2ZKBDIN	06C2
		2ZKEYFIL	0010
		2ZKEYPM	001A
		2ZLDATA	0002
		2ZLDYTE	0582
		2ZLFETBL	067E
		2ZLSTBR	073D
		2ZLSTDM	0721
		2ZM0	0020
		2ZM1	0020
		2ZMAINB0	009E
		2ZMAINB1	00A0
		2ZMDJNZ	002D
		2ZMEMHI	0035
		2ZMPUSL	0040
		2ZNEG1	FFFF
		2ZNOVALS	0004
		2ZNRPTS	003D
		2ZORPG0	0100
		2ZORPG1	01FD
		2ZOUTCLR	0102
		2ZOUTMSG	0104
		2ZOV3B1	0311
		2ZPBRK	0019
		2ZPDIGIT	000E
		2ZPNT2	0108
		2ZPSEGH1	0000

APPENDIX C COMMAND SUMMARY

The following is a summary of the commands implemented by the HSE-49 emulator monitor. Within each command group, tokens in each column indicate options the user has when invoking those commands.

Tokens in square brackets indicate dedicated keys on the keyboard (some keys having shared functions); angle brackets enclose hex digit strings used to specify an address or data parameter. Parameters in parentheses are optional, with the effects explained above. The notation used is as follows:

<SMA> — Starting Memory Address for block command,
<EMA> — Ending Memory Address for block command,
<LOC> — LOcation for individual accesses,
<DATA> — DATA byte.

Asterisks (*) indicate the default condition for each command; thus that token is optional and serves to regularize the command syntax.

Program/data entry and verification commands:

[EXAM] [PROG MEM]* <LOC> [.] [NEXT]
[DATA MEM] [PREV]
[REGISTER] [.]
[HWRE REG]
[PROG BRK]
[DATA BRK]

Program/data initialization commands:

[FILL] [PROG MEM]* <SMA> [.] <EMA> [.] <DATA> [.]
[DATA MEM]
[REGISTER]
[HWRE REG]
[PROG BRK]
[DATA BRK]

Intellec® development system or TTY interface commands (for transferring HEX format files):

[UPLOAD] [PROG MEM]* <SMA> [.] <EMA> [.]
[DATA MEM]
[REGISTER]
[HWRE REG]
[PROG BRK]
[DATA BRK]
[DNLOAD] [PROG MEM]* [.]
[DATA MEM]
[REGISTER]
[HWRE REG]
[PROG BRK]
[DATA BRK]

Formatted data dump to TTY or CRT:

[LIST] [PROG MEM]* <SMA> [.] <EMA> [.]
[DATA MEM]
[REGISTER]
[HWRE REG]
[PROG BRK]
[DATA BRK]

Program execution commands:

[GO] [NO BREAK]* (<SMA>) [.]
[W/ BREAK] [.]
[SING STP]
[AUTO BRK]
[AUTO STP]
[GO/RST] [NO BREAK]* [.]
[W/ BREAK]
[SING STP]
[AUTO BRK]
[AUTO STP]

Breakpoint setting and clearing:

[SET BRK] [PROG MEM]* <LOC> [.] <LOC> ... [.]
[DATA MEM]
[CLR BRK] [PROG MEM]* <LOC> [.] <LOC> ... [.]
[DATA MEM]

APPENDIX D ERROR MESSAGES

The following error message codes are used by the monitor software to report an operator or hardware error. Errors may be cleared by pressing [CLR/PREV] or [END/]. The format used for reporting errors is "Error—n" where "n" is a hex digit.

Operator Errors

1. Illegal command initiator.
2. Illegal command modifier or parameter digit.
3. Illegal terminator for Examine command.
4. Illegal attempt to clear Error mode.
- 5-9. Not used.

Hardware Errors

- A. ASCII error — non-hex digit encountered in data field of hex format record.
- B. Breakpoint error. Break logic activated though breakpoints not enabled.
- C. Hex format record checksum error. Note — the checksum will not be verified if the first character of the checksum field is a question mark ("?") rather than a hexadecimal digit. This allows object files to be patched using the ISIS text editor without the necessity of manually recomputing the checksum value.
- D. Not used.
- E. Execution processor failed to respond to a command or parameter passed to it by the master processor. EP automatically reset. EP internal status may be lost. Program memory not affected.
- F. Not used.

entire line at a time. Line printers are usually a bit faster than character printers, but they usually don't offer the print quality of character printers.

In recent years, the "computer boom" has caused the price of printers to tumble markedly. High volume production competition, and the tremendous demand for reliable print mechanisms have all contributed to the decrease in price. Because of their simplicity, line printer mechanisms have decreased in price faster than other mechanisms. Therefore, when high quality print is not needed, a line printer is a very attractive choice.

This application note describes how to control an 80 column impact-line printer with an 8049. The complete software listing is included in the appendix.

The 8049 is the high-performance member of the MCS-48™ microcontroller family. The Processor has all of the features of the 8048 plus twice the amount of program and data memory and an 11MHz clock speed. For details about the 8049, please refer to the M-248 user's manual.

II. PRINT MECHANISM DESCRIPTION

The model 820 printer is available from CATHON ELECTRONICS (3001 BERTHOVEN STREET, LOS ANGELES, CA 90008). This inexpensive and simple printer is ideal for applications requiring 80 columns of dot matrix alpha-numeric information.

The model 820 printer is comprised of three basic sub-assemblies: the chassis or frame, the paper feed mechanism, and the print head. The chassis in Figure 2-1 gives the physical dimensions of the basic print mechanism. The basic chassis for the printer is constructed out of four sheet metal stampings. These stampings are screwed together to form a sturdy base on which all other components of the printer are mounted.

The paper feed mechanism consists of a footed wheel, a solenoid, a tension spring, and a "catcher". When the solenoid is activated, the arm of the solenoid pulls against the spring and drags over the footed wheel. When the solenoid is released, its arm is pulled by the spring, but this time the arm grabs a tooth on the wheel and pulls the wheel forward which advances the paper. A "catcher", which is merely a piece of plastic held against the footed wheel, is added to ensure the paper is advanced only one "tooth" position. When the solenoid is activated.

The head is comprised of seven solenoids which are mounted in a common housing. The solenoids are activated in a circle, but their hammer arms are staggered along the vertical axis. These staggered hammer arms are the all-important

USING THE 8049 AS AN 80 COLUMN PRINTER CONTROLLER

INTRODUCTION

1.1. Application note details using INTEL's 8049 microcomputer as a dot matrix printer controller. It provides INTEL application notes, (e.g. AP-21 and AP-24) described using intelligent processors and peripherals to control large printer mechanisms. This application note expands upon the theme established in these prior notes and extends the concept to include a complete functional 80 column printer using a single line buffer. For convenience this application note is divided into six sections:

1. INTRODUCTION
2. PRINT MECHANISM DESCRIPTION
3. INTERNAL CIRCUITRY
4. SOFTWARE
5. CONCLUSION
6. APPENDIX

Over the last few years 80 column output devices have become somewhat of a de facto output standard for business and scientific data processing applications. It should be mentioned that by no means is the 80 column format a "new" standard. 80 column computer cards have been around for more than 20 years and the existence of these cards in the early days of computing was the 80 column format is a standard today.

Many CRT displays use the 80 by 14 format and most computer terminals use the same. One reason for this is that 80 columns of 12 pitch text is standard 8.5 inch by 11 inch paper exactly. It is an entire line and a half ample room for margins. 80 column format is a standard for scientific and business data processing.

Printers are used in a variety of ways, for impact or non-impact printing. Impact printing is the most common and a character or word is printed by striking the paper with a hammer. Non-impact printing is the most expensive and a character or word is printed by using a laser or other type of printer. The type of printer is determined by the mechanical motion of the hammer on the paper. The type of printer is determined by the mechanical motion of the hammer on the paper. The type of printer is determined by the mechanical motion of the hammer on the paper.

Printer printing is capable of printing one character at a time. The printer home typewriter is a character printer. The printer must print one character at a time.

Using the 8049 as an 80 Column
Printer Controller

John Ketausky
Applications Engineering

USING THE 8049 AS AN 80 COLUMN PRINTER CONTROLLER

I. INTRODUCTION

This Application Note details using INTEL's 8049 microcomputer as a dot matrix printer controller. Previous INTEL Application notes, (e.g. AP-27 and AP-54) described using intelligent processors and peripherals to control single printer mechanisms. This Application note expands upon the theme established in these prior notes and extends the concept to include a complete bi-directional 80 column printer using a single line buffer. For convenience this application note is divided into six sections:

1. INTRODUCTION
2. PRINT MECHANISM DESCRIPTION
3. INTERFACE CIRCUITRY
4. SOFTWARE
5. CONCLUSION
6. APPENDIX

Over the last few years 80 column output devices have become somewhat of a defacto output standard for business and some data processing applications. It should be mentioned that by no means is the 80 column format a "new" standard. 80 column computer cards have been around for more than 20 years and perhaps the existence of these cards in the early days of computers is why the 80 column format is a standard today.

Many CRT terminals use the 80 by N format and to complement this a number of printers use this same format. One reason, aside from those historic in nature, for the 80 column standard is that 80 columns of 12 pitch text on standard typewritten 8.5 inch by 11 inch paper completely fills up an entire line and allow ample room for margins. So, the 80 column format is an aesthetically convenient format.

Printers are usually divided into either impact or non-impact and a character or line oriented device. Impact printers actually use some type of "striker" to place ink on the paper. More often than not the ink is contained on a ribbon which is placed between the striker and the paper. Non-impact printers use some means other than direct pressure to place the characters on the paper. This type of printer is very fast because there is very little mechanical motion associated with placing the characters on the paper. However, because the paper is required to be treated with a special substance, it is not as convenient as an impact printer.

Character printers are capable of printing one character at a time. (Any standard home typewriter is in effect a character printer.) Line printers must print an

entire line at a time. Line printers are usually quite a bit faster than character printers, but they usually don't offer the print quality of character printers.

In recent years, the "computer boom" has caused the price of printers to tumble markedly. High volume production, competition, and the tremendous demand for reliable print mechanisms have all contributed to the decrease in price. Because of their simplicity, line printer mechanisms have decreased in price faster than other mechanisms. Therefore, when high quality print is not needed, a line printer is a very attractive choice.

This application note describes how to control an 80 column impact-line printer with an 8049/8039. The complete software listing is included in the appendix. The 8049 is the high-performance member of the MCS-48™ microcontroller family. The Processor has all of the features of the 8048 plus twice the amount of program and data memory and an 11MHz clock speed. For details about the 8049, please refer to the MCS-48 user's manual.

II. PRINT MECHANISM DESCRIPTION

The model 820 printer is available from C. ITOH ELECTRONICS (5301 BEETHOVEN STREET, LOS ANGELES, CA 90066). This inexpensive and simple printer is ideal for applications requiring 80 columns of dot matrix alpha-numeric information.

The model 820 printer is comprised of three basic sub-assemblies; the chassis or frame, the paper feed mechanism, and the print head. The diagram in Figure 2.1 gives the physical dimensions of the basic print mechanism. The basic chassis for the printer is constructed out of four sheet metal stampings. These stampings are screwed together to form a sturdy base on which all other components of the printer are mounted.

The paper feed mechanism consists of a toothed wheel, a solenoid, a tension spring, and a "catcher." When the solenoid is activated, the arm of the solenoid pulls against the spring and drags over the toothed wheel. When the solenoid is released, its arm is pulled by the spring, but this time the arm grabs a tooth on the wheel and pulls the wheel forward which advances the paper. A "catcher," which is merely a piece of plastic held against the toothed wheel, is added to assure that the paper is advanced only one "tooth" position each time the solenoid is activated.

The print head is comprised of seven solenoids which are mounted in a common housing. The solenoids are physically mounted in a circle, but their hammers are positioned linearly along the vertical axis. These seven vertically positioned hammers are the strikers that actually do the printing.

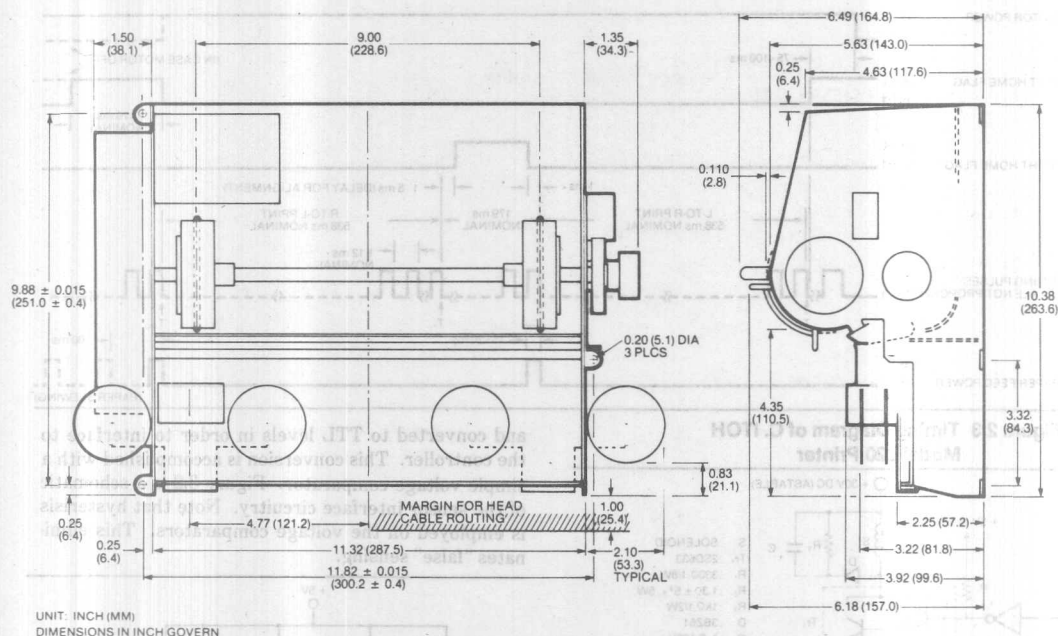


Figure 2.1 Physical Dimensions of C. ITOH Model 820 Printer

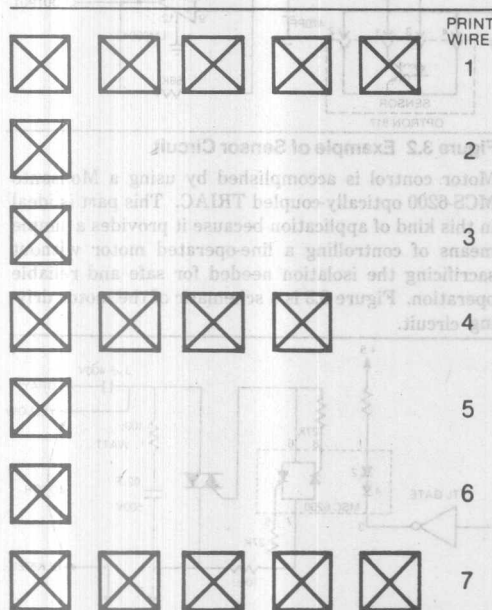


Figure 2.2 "Formation" of a Character by a Dot Matrix Printer

A motor, mounted toward the back of the print mechanism, drives a rubber toothed belt which turns a roller guide. A motor turns a guide that moves the print head from right to left and left to right. By properly timing the current flow through the solenoids while the print head is moving across the paper, characters can be formed. Figure 2.2 illustrates how the dot matrix printer "forms" its characters.

The timing pulses for the print head mechanism are generated by an opto-electronic sensor. This sensor, located on the left side plate of the printer, informs the print controller when to apply current to the print head mechanism. This "on-board timing wheel" assures that all characters will be properly spaced and that they will all be "in-line" in a vertical sense.

The print mechanism is also equipped with two additional sensors. These are the left home position sensor, located near the left front of the mechanism, and the right home position sensor, located near the right front of the print mechanism. These sensors simply tell the controller when the print head is in either the left or right home position. A complete timing chart for the printer is shown in Figure 2.3.

III. INTERFACE CIRCUITRY

The manual supplied with the printer recommends some specific interface circuitry. For the most part the circuitry used in this Application Note followed these suggestions. The circuitry needed to drive the print head solenoid is shown in Figure 3.1. This same

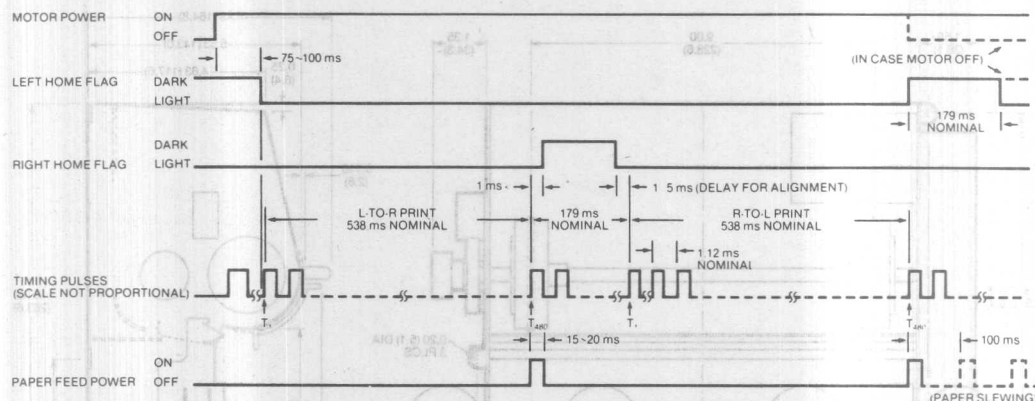


Figure 2.3 Timing Diagram of C. ITOH Model 820 Printer

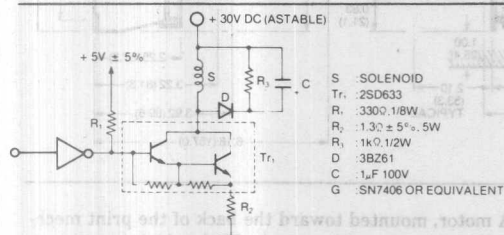


Figure 3.1 Solenoid Drive Circuit (Eliminate R2 for Line Feed Solenoid)

circuit is used to drive the line feed solenoid except that the current limiting resistor R2 is eliminated. This resistor is not needed because the line feed solenoid is physically much larger than the print head solenoids and can tolerate much higher levels of current. The print head drivers are connected to an 8212 latch. The latch is interfaced to the BUS PORT on the 8049 and is enabled whenever the WR pin and the BIT 4 of PORT 1 are coincidentally low. The line feed driver is connected to PORT 1 BIT 1 of the 8049.

Note that the driver is simply a Darlington transistor that is driven by an open collector TTL gate. Resistor R2 is the current limiting resistor and diode D, capacitor C, and resistor R3 are used to "dampen" the inductive spike that occurs when driving solenoid S. This circuit is repeated for each of the seven solenoids in the print head. It should be mentioned that, although the type of Darlington transistor needed to drive the print head is not critical, a collector current rating of at least 5 amps and a breakdown voltage (Vce) of at least 100 volts is needed. Transistors that do not meet these requirements will be damaged by the inductive kickback of the solenoids.

As mentioned in Section 2, the printer provides some sensor interface signals that are derived via three optoelectronic sensors. These signals must be amplified

and converted to TTL levels in order to interface to the controller. This conversion is accomplished with a simple voltage comparator. Figure 3.2 is a schematic of the sensor interface circuitry. Note that hysteresis is employed on the voltage comparators. This eliminates "false" sensing.

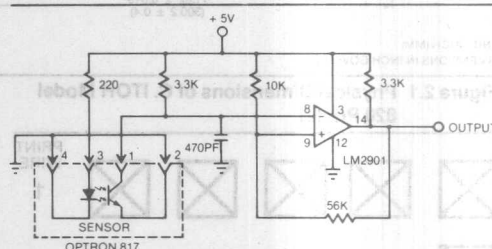


Figure 3.2 Example of Sensor Circuit

Motor control is accomplished by using a Monsanto MCS-6200 optically-coupled TRIAC. This part is ideal in this kind of application because it provides a simple means of controlling a line-operated motor without sacrificing the isolation needed for safe and reliable operation. Figure 3.3 is a schematic of the motor driving circuit.

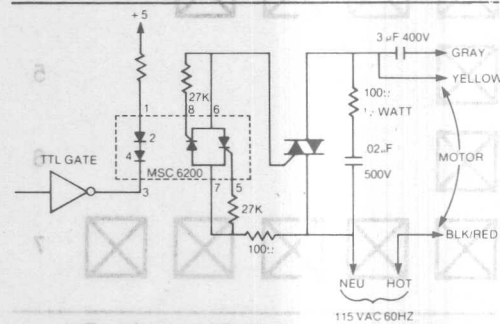


Figure 3.3 Motor Driving Circuit

To interface 8049 to the outside world one 8212 latch was used. This latch was connected to the BUS PORT and is enabled by an INS or MOVX instruction coincident with BIT 4 of PORT 1 being in a logical zero state. In this configuration, the 8212 was used to hold the data until read by the 8049. The connection of the 8212 to the 8049 is shown in Figure 3.4 and the parallel port timing diagram is shown in Figure 3.5. The 8212 parallel port was connected to the LINE PRINTER OUTPUT of an INTELLEC MICROCOMPUTER DEVELOPMENT SYSTEM.

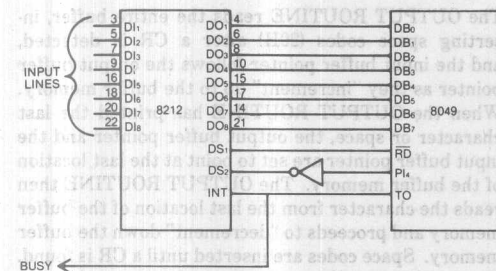


Figure 3.4 Connection of the 8212 Input Port to the 8049

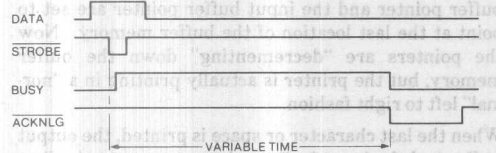


Figure 3.5 Parallel Port Timing

IV. SOFTWARE

As mentioned in Section 2, the bulk of the timing needed to control the printer is actually generated by the printer itself. Therefore, all the software must do is harness these timing signals and turn on and off the right solenoids at the right time.

To make things easy, the software needed to drive the printer is broken into four separate routines. These are:

1. INITIALIZATION ROUTINE
2. INPUT ROUTINE
3. OUTPUT ROUTINE
4. LOOKUP ROUTINE

The INITIALIZATION ROUTINE turns the motor on and checks the opto-electronic sensors. If a failure is found, the routine turns off the motor and loops on itself. This insures that the print mechanism is cycled properly before characters are accepted for printing.

This routine also initializes all of the variables used by the printer.

The INPUT ROUTINE reads the characters that are present in the 8212 input port and writes them into the 8049's buffer memory. The routine then checks the characters to see if a CARRIAGE RETURN (ASCII OCH) has been transmitted. If a CR is detected, the input routine automatically inserts a LINE FEED as the next character. When the input routine detects a LINE FEED, it stops reading characters and sets the direction bits and the print bit in the status register. This action evokes the OUTPUT ROUTINE. A detailed flowchart of the INPUT ROUTINE is shown in Figure 4.1.

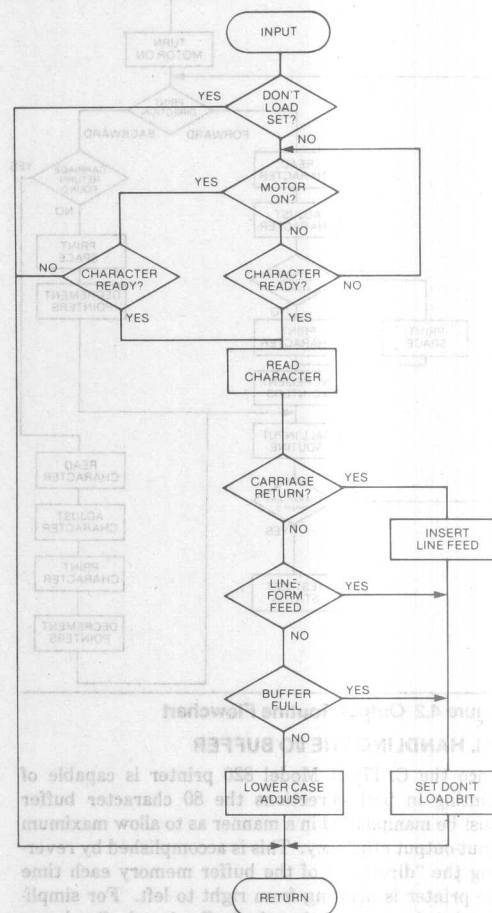


Figure 4.1 Input Routine Flowchart

The OUTPUT ROUTINE initializes both the input and output buffer pointers and then reads the characters from the 8049's buffer memory. After a character is read the OUTPUT ROUTINE calls the LOOKUP ROUTINE which reads the proper bit pattern to form that character. This bit pattern is then used to strobe the solenoids. After each character is printed, the OUTPUT ROUTINE calls the INPUT ROUTINE and another character is placed into the buffer memory. This type of operation guarantees that the input buffer cannot "overflow" the output buffer. A flowchart of the OUTPUT ROUTINE is shown in Figure 4.2.

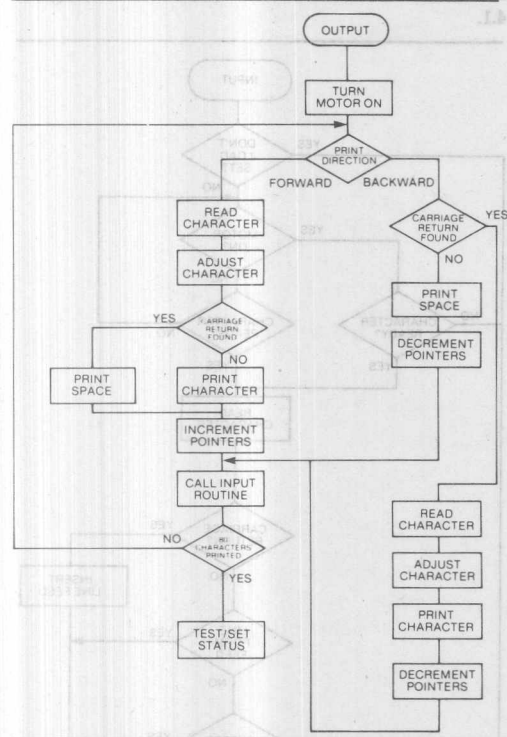


Figure 4.2 Output Routine Flowchart

IV-I. HANDLING THE I/O BUFFER

Since the C. ITOH Model 820 printer is capable of printing in both directions the 80 character buffer must be manipulated in a manner as to allow maximum input-output efficiency. This is accomplished by reversing the "direction" of the buffer memory each time the printer is printing from right to left. For simplicity, if it is assumed that the buffer is only five bytes long, Figure 4.3 can be used to help explain the buffer operation.

Initially the input buffer pointer is loaded with the address of the first location in the buffer memory. As characters are read, the input buffer pointer increments and fills the buffer memory as shown in Figure 4.3(b) through 4.3(f). When a CARRIAGE RETURN-LINE FEED (CRLF) is encountered the input buffer pointer and the output buffer pointer are reset back to the first location. The OUTPUT ROUTINE then reads the character from the first location in the buffer memory, increments the output buffer pointer and calls the INPUT ROUTINE, which reads another character from the parallel input port.

The OUTPUT ROUTINE reads the entire buffer, inserting space codes (20H) after a CR is detected, and the input buffer pointer follows the output buffer pointer as they "increment" up to the buffer memory. When the OUTPUT ROUTINE has printed the last character or space, the output buffer pointer and the input buffer pointer are set to point at the last location of the buffer memory. The OUTPUT ROUTINE then reads the character from the last location of the buffer memory and proceeds to "decrement" down the buffer memory. Space codes are inserted until a CR is found. Figure 4.3(1) to 4.3(0).

The input buffer pointer follows the output buffer pointer just as in the previous case. When the last, or in this case the first character is printed, the output buffer pointer and the input buffer pointer are set to point at the last location of the buffer memory. Now the pointers are "decrementing" down the buffer memory, but the printer is actually printing in a "normal" left to right fashion.

When the last character or space is printed, the output buffer and the input buffer pointer are set to the first location of the buffer memory and printing takes place in a reverse or right to left manner. After this line is printed, the print head and both buffer pointers are in the same position as they were initially. So, four lines must be printed before the buffer pointers and the print head complete a cycle. Each of these situations is handled separately by four different sub-routines: CASE0, CASE1, CASE2, and CASE3.

IV-II. TIMING

All critical timing for the printer controller came from two basic sources; the timing sensors on the printer and the internal eight-bit timer of the 8049.

The internal timer of the 8049 was used to control the length of time the solenoids were fired (600 microseconds) and was also used as a "one-shot" to align the printer. This alignment is needed to make the "backward" printing line up vertically with the normal or forward printing. The "one-shot" is used to measure the time from the last column of the last character position until the right sensor flag is covered.

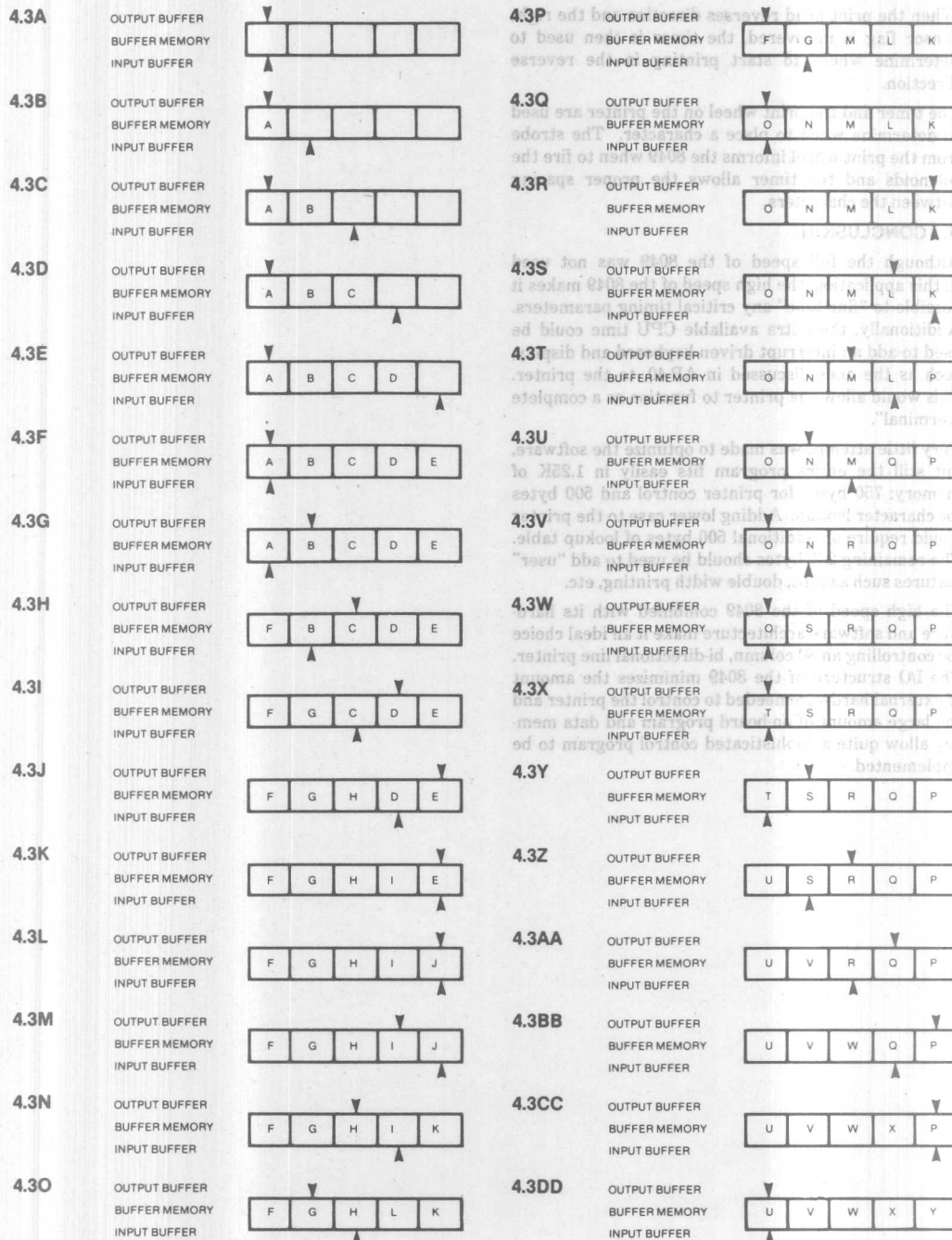


Figure 4.3 I/O Buffer Handler

When the print head reverses direction and the right sensor flag is uncovered, the timer is then used to determine where to start printing in the reverse direction.

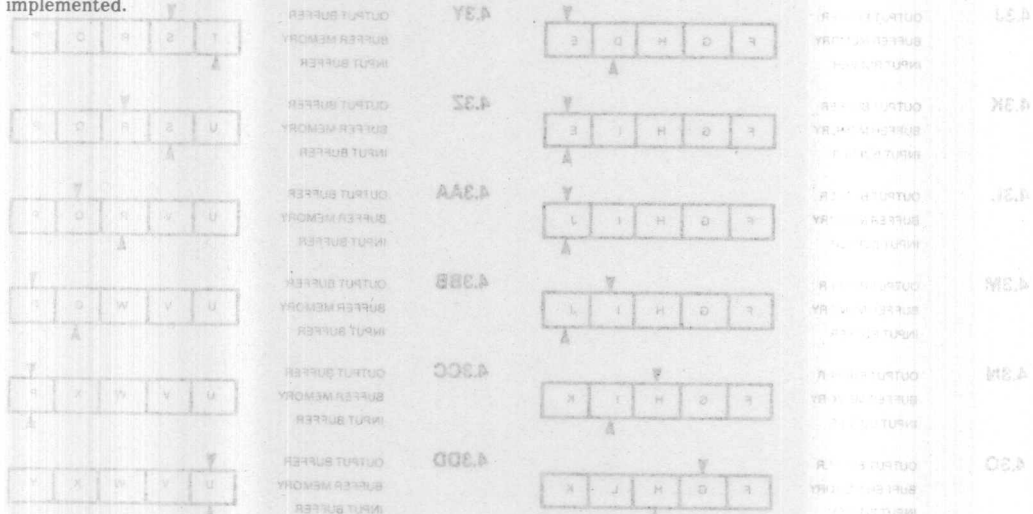
The timer and the print wheel on the printer are used to determine when to place a character. The strobe from the print wheel informs the 8049 when to fire the solenoids and the timer allows the proper spacing between the characters.

V. CONCLUSION

Although the full speed of the 8049 was not used in this application, the high speed of the 8049 makes it possible to "fine-tune" any critical timing parameters. Additionally, the extra available CPU time could be used to add an interrupt driven keyboard and display, such as the ones discussed in AP-40, to the printer. This would allow the printer to function as a complete "terminal".

Very little attempt was made to optimize the software, but still the entire program fits easily in 1.25K of memory; 750 bytes for printer control and 500 bytes for character lookup. Adding lower case to the printer would require an additional 500 bytes of lookup table. The remaining 250 bytes should be used to add "user" features such as tabs, double width printing, etc.

The high-speed of the 8049 combined with its hardware and software architecture make it an ideal choice for controlling an 80 column, bi-directional line printer. The I/O structure of the 8049 minimizes the amount of external hardware needed to control the printer and the large amount of on-board program and data memory allow quite a sophisticated control program to be implemented.



Inte



APPENDIX B. MONITOR LISTING

LDC OBJ

SEQ

SOURCE STATEMENT

0000
0001
0002
0003
0004
0005

0006
0007
000F
0020

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43

.....

THIS PROGRAM IMPLEMENTS CONTROL OF THE C. ITOM MODEL 82B
 PPINTER THE HARDWARE CONFIGURATION IS AS SUCH:
 :B212 INPUT PORT ON BUS = DATA INPUT
 :B212 OUTPUT PORT ON BUS = OUTPUT TO SOLENOID HAMMERS
 :T1 INPUT = CHARACTER POSITIONING SENSOR ON PRINTER
 :TB INPUT = INTERRUPT FROM 8212 INPUT PORT
 :PORT 10 = MOTOR ON, LOW = ON
 :PORT 11 = LINE FEED STROBE, LOW = ON
 :PORT 16 = LEFT MARGIN SENSOR, LOW WHEN COVERED, HIGH WHEN OPEN
 :PORT 17 = RIGHT MARGIN SENSOR, LOW WHEN COVERED, HIGH WHEN OPEN
 :T1 = PIN 2 OF LM339, PRINT WHEEL SENSOR
 :PORT 16 = PIN 13 OF LM339
 :PORT 17 = PIN 14 OF LM339

.....

SYSTEM EQUATES

:POINTS AT INPUT LOCATION
 :POINTS AT OUTPUT LOCATION
 :STATUS FOR PRINTING
 :STROBE COUNTER

:BIT 0 = LINE FEED SET
 :BIT 1 = PRINT
 :BIT 2 = CONTINUE
 :BIT 3 = CR FOUND
 :BIT 4 = LF FOUND
 :BIT 5 = LF FOUND IN PRINTING
 :BIT 6 = PRINT DIRECTION
 :0 = RIGHT TO LEFT
 :1 = LEFT TO RIGHT
 :BIT 7 = BUFFER LOAD DIRECTION
 :0 = FIRST TO MAX
 :1 = MAX TO FIRST
 :THE LINE COUNTER

:MAX BUFFER LOCATION
 :BOTTOM OF BUFFER

LOC	OBJ	SEQ	SOURCE STATEMENT	INSTRATE	33R009	LOC	OBJ
		44	GETCABAND INT TSD	JUSTUOS.A	VON	022AD	1
0000		45	INTERRUPT ORIGIN 0000H	THATRT	JJAO	022AD	1
		46	GETCABAND INT TSD	JUSTUOS.A	VON	022AD	1
		47	JUMP OVER THE INTERRUPT LOCATIONS	TSTRT	JJAO	022AD	1
		48	SETUP STATUS	TSTRT	JJAO	022AD	1
0000 15		49	DISE INTERRUPT	DO NOT USE INTERRUPTS		022AD	1
0001 0400		50	THIR JUMP TO BEGIN	DO NOT BEGIN THE PROGRAM		022AD	1
		51	DO NOT A THIR 001	THATRT	JJAO	022AD	1
000A		52	ORG 0A00H	022AD	022AD	022AD	1
		53	DO NOT THIR INT TSD	JUSTUOS.A	VON	022AD	1
		54	GETCABAND START THE PROGRAM	THATRT	JJAO	022AD	1
		55	DO NOT THIR 001	THATRT	JJAO	022AD	1
		56	DO NOT LOOP UNTIL THE BUFFER FILLS UP	022AD	022AD	022AD	1
		57	GETCABAND INT TSD	JUSTUOS.A	VON	022AD	1
000A FD		58	PRINT MOV 0A00H A STATUS	DO NOT GET THE STATUS		022AD	1
0000 3211		59	DO NOT JUMP TO PRINT	IF PRINTING, CONTINUE		022AD	1
0000 3400		60	CALL TLOBUF	DO NOT READ INTO THE BUFFER		022AD	1
000F 040A		61	JMP PRINT	DO NOT LOOP		022AD	1
		62				022AD	1
		63	THIS ROUTINE PRINTS A LINE			022AD	1
		64	DO NOT THIR 001	DO NOT FIRST SAVES THE STATUS		022AD	1
		65	AND THEN DETERMINES WHICH DIRECTION TO PRINT			022AD	1
		66	AND HOW TO MANIPULATE THE BUFFER			022AD	1
		67	GETCABAND INT TSD	JUSTUOS.A	VON	022AD	1
0011 0409		68	LPRT: JUMP TO STACK	DO NOT FIX UP THE STATUS		022AD	1
0013 F224		69	LPRT: JB7 CASE23	DO NOT JUMP TO CASE 2 AND 3		022AD	1
0015 0417		70	DO NOT JUMP TO CASE B1	DO NOT JUMP TO CASE B AND 1		022AD	1
		71	DO NOT A THIR 001	THATRT	JJAO	022AD	1
		72	GETCABAND CASE B1: LOADING THE BUFFER FROM FIRST TO MAX			022AD	1
		73	DO NOT THIR 001	THATRT	JJAO	022AD	1
0017 0920		74	CASE B1: MOV 0A00H OUTBUF.#FIRST	DO NOT SET UP OUTBUF		022AD	1
0019 0028		75	MOV INBUF.#FIRST	DO NOT SET UP INBUF		022AD	1
0010 FA		76	THIR 0A00H INT MOV 0A00H A SAVPNT	DO NOT GET THE SAVED STATUS		022AD	1
0010 940C		77	DO NOT THIR 0A00H INT CALL 0A00H MOTON	DO NOT TURN ON THE MOTOR		022AD	1
001E D252		78	DO NOT JB6 THIR CASE1	DO NOT PRINT FORWARD		022AD	1
0020 0403		79	DO NOT CALL INT PRINTBK	DO NOT GET READY TO PRINT BACKWARDS		022AD	1
0022 0431		80	DO NOT THIR JUMP TO CASE B	DO NOT PRINT BACKWARDS		022AD	1
		81	GETCABAND THIR INT TSD	JUSTUOS.A	VON	022AD	1
		82	CASE23: LOADING BUFFER FROM MAX TO FIRST			022AD	1
		83	DO NOT 0001	022AD	022AD	022AD	1
0024 096F		84	CASE23: MOV OUTBUF.#MAX	DO NOT SET UP OUTBUF		022AD	1
0026 006F		85	MOV INBUF.#MAX	DO NOT SET UP INBUF		022AD	1
0028 FA		86	MOV A SAVPNT	DO NOT GET THE PRINT STATUS		022AD	1
0029 940C		87	CALL MOTON	DO NOT TURN ON THE MOTOR		022AD	1
0020 D2C2		88	JB6 CASE3	DO NOT PRINT LEFT TO RIGHT		022AD	1
0020 0403		89	CALL PRINTBK	DO NOT GET READY TO PRINT BACKWARDS		022AD	1
002F 0400		90	JMP CASE2	DO NOT PRINT RIGHT TO LEFT		022AD	1
		91				022AD	1
		92	EJECT			022AD	1

LOC	OBJ	SEQ	SOURCE STATEMENT	THIRDTIME 238002	91	140	312
0031	F1	93	CASE1: MOV A, @OUTBUF ;GET THE CHARACTER				
0032	3491	94	CALL FXPRNT ;ADJUST FOR PRINTING				
0034	B120	95	MOV @OUTBUF, @2BH ;PUT A SPACE IN BUFFER RAM				
0036	F242	96	JB7 ZND FDC ;FOUND A CR				
0038	945E	97	CALL INCTST ;UPDATE OUTBUF				
003A	C6AE	98	JZ WATCHD ;WAIT FOR END				
003C	BF20	99	MOV JUNK1, @2BH ;GET A SPACE TO PRINT				
003E	9463	100	CALL GTPRNT ;GO PRINT A SPACE				
0040	B431	101	JMP CASE0 ;LOOP				
0042	BF20	102	MOV JUNK1, @2BH ;GO PRINT THE LAST SPACE				
0044	9463	103	FDC1: CALL GTPRNT ;GO PRINT A CHARACTER				
0046	945E	104	CALL INCTST ;CHECK OUT BUFFER				
0048	C6AE	105	JZ WATCHD ;WAIT FOR THE END				
004A	F1	106	MOV A, @OUTBUF ;GET THE CHARACTER				
004B	B120	107	MOV @OUTBUF, @2BH ;PUT A SPACE THERE				
004D	3491	108	CALL FXPRNT ;FIX THE CHARACTER UP				
004F	AF	109	MOV JUNK1, A ;SAVE IT				
0050	B444	110	JMP FDC1 ;LOOP				
		111	;				
		112	;				
		113	;CASE 1, PRINTING LEFT TO RIGHT, LOADING BUFFER FROM				
		114	;FIRST TO MAXIMUM DEFINED BY THE ACC				
		115	;				
0052	F1	116	CASE1: MOV A, @OUTBUF ;GET THE CHARACTER				
0053	3491	117	CALL FXPRNT ;ADJUST FOR PRINTING				
0055	AF	118	MOV JUNK1, A ;SAVE ACC				
0056	B120	119	MOV @OUTBUF, @2BH ;PUT A SPACE IN THE BUFFER				
0058	F262	120	JB7 CRFOND ;FOUND A CR?				
005A	9463	121	CALL GTPRNT ;GO PRINT THE CHARACTER				
005C	945E	122	CALL INCTST ;CHECK THE BUFFER				
005E	C675	123	JZ WATCHD ;IS THE LAST CHARACTER BEING PRINTED?				
0060	B452	124	JMP CASE1 ;LOOP				
0062	B120	125	CRFOND: MOV @OUTBUF, @2BH ;PUT A SPACE IN THE BUFFER MEMORY				
0064	BF20	126	MOV @JUNK1, @2BH ;PUT A SPACE IN TEMP LOCATION				
0066	9463	127	CALL GTPRNT ;GO PRINT THE SPACE				
0068	945E	128	CALL INCTST ;CHECK THE BUFFER				
006A	C675	129	JZ WATCHD ;LAST CHARACTER PRINTED?				
006C	F1	130	MOV A, @OUTBUF ;GET THE NEXT CHARACTER				
006D	3491	131	CALL FXPRNT ;ADJUST IT				
006F	B462	132	JMP CRFOND ;LOOP				
		133	\$EJECT				

LDC	OBJ	SEQ	SOURCE STATEMENT	THIRDTATE 334002	631	L20
		134	;			
		135	THIS ROUTINE CALLS THE LINE FEED			
		136	;			
0071	9478	137	DOLF: CALL LINEFD ;STROBE LINE FEED SOLENOID			
0073	048A	138	JMP CASE2 ;GO BACK TO THE PRINT ROUTINE			
		139	;			
		140	THIS ROUTINE COMPLETES A LINE WHEN THE PRINT			
		141	HEAD IS MOVING LEFT TO RIGHT			
		142	;			
0075	27	143	WATCH: CLRNT WAIT ;ZERO ACC			
0076	62	144	MOVIE TET A ;ZERO TIMER			
0077	55	145	STRT TET A ;START THE TIMER			
0078	348B	146	CALL LDBUF ;GO READ THE LAST CHARACTER			
007A	B9	147	LDOPM: ;EXAMIN PORT ONE			
007B	F27A	148	JB7: J11 ALDOPM ;CHECK RIGHT HAND SENSOR			
007D	65	149	STOP TICT ;STOP THE TIMER			
007E	FD	150	MOV A, STATUS ;GET THE STATUS			
007F	5205	151	JB2: J11 ;JUMP IF CONTINUE IS SET			
0081	94DF	152	CALL MOTOF ;TURN MOTOR OFF			
0083	53FD	153	ANL A, 00FDH ;RESET BIT ONE			
0085	53FB	154	ORL A, 00FBH ;RESET CONTINUE BIT			
0087	AD	155	MOV STATUS, A ;RESTORE STATUS			
0088	FA	156	MOVAND A, SAVPNT ;GET THE SAVED STATUS			
0089	B271	157	JB5: J11 DOLF ;DO A LINE FEED IF BIT IS SET			
008B	048A	158	JMP: J11 PRNT ;GO BACK TO PRINT ROUTINE			
		159	;			
		160	;			
		161	CASE 2: PRINTING RIGHT TO LEFT. LOADING BUFFER FROM			
		162	;			
		163	;			
		164	;			
008D	F1	165	CASE2: MOV A, 00OUTBUF ;GET THE CHARACTER			
008E	3491	166	CALL FXPRNT ;ADJUST FOR PRINTING			
0090	B12B	167	MOV 00OUTBUF, 02BH ;PUT A SPACE IN BUFFER RAM			
0092	F29E	168	JB7: J11 FDCR ;FIND A CR YET			
0094	9472	169	CALL DECTST ;CHECK THE BUFFER			
0096	C6AE	170	JZ WATCHD ;IF ZERO WAIT FOR SENSOR FLAG			
0098	BF2B	171	MOV JUNK1, 02BH ;PUT SPACE IN TEMP LOCATION			
009A	9463	172	CALL GTPRNT ;GO PRINT SPACE			
009C	048D	173	JMP CASE2 ;LOOP			
009E	BF2B	174	FDCR: MOV JUNK1, 02BH ;GET A SPACE			
00A0	9463	175	FDCR1: CALL GTPRNT ;GO PRINT THE CHARACTER			
00A2	9472	176	CALL DECTST ;CHECK THE BUFFER			
00A4	C6AE	177	JZ WATCHD ;LEAVE IF DONE			
00A6	F1	178	MOV A, 00OUTBUF ;GET A CHARACTER			
00A7	3491	179	CALL FXPRNT ;ADJUST THE CHARACTER FOR PRINTING			
00A9	AF	180	MOV JUNK1, A ;SAVE IT			
00AA	B12B	181	MOV 00OUTBUF, 02BH ;PUT A SPACE WHERE THE CHARACTER WAS			
00AC	04AB	182	JMP FDCR1 ;LOOP			
		183	#EJECT			

LOC	OBJ	SEQ	SOURCE STATEMENT	THANETATS 338052	97	100 301
		184	;			
		185	;THIS ROUTINE WAITS FOR THE SENSOR FLAGS TO BE COVERED			
		186	;WHEN PRINTING RIGHT TO LEFT			
		187	CALL LDBUF			
BBAE 3400		188	WATCHD: CALL LDBUF			
BBB8 09		189	IN A,P1			
BBB1 D2AE		190	JB6 WATCHD: LOOP IF SENSOR IS NOT COVERED			
BBB3 FD		191	MOV A,STATUS			
BBB4 52BA		192	JB2 OVR			
BBB6 94DF		193	CALL MOTOFF			
BBB8 53FD		194	ANL A,0BF0H			
BBBA 53FB		195	OVR: ANL A,0BF0H			
BBBC AD		196	MOV A,STATUS			
BBBD FA		197	MOV A,SAVPNT			
BBBE B271		198	JB5 DOLF			
BBCB B4BA		199	JMP PRNT			
		200	;			
		201	;			
		202	;			
		203	;			
BBB2 F1		204	CASE3: MOV A,0			
BBB3 3491		205	CALL FXPRNT			
BBB5 AF		206	MOV A,0			
BBB6 B120		207	MOV A,0			
BBB8 F2D2		208	JB7 CRFND			
BBBA 9463		209	CALL GTPRNT			
BBBC 9472		210	CALL DECTST			
BBBE C675		211	JZ WATCHD: LOOP IF DONE			
BBB8 B4C2		212	JMP CASE3			
BBB2 B120		213	CRFND: MOV A,0			
BBB4 BF20		214	MOV A,0			
BBB6 9463		215	CALL GTPRNT			
BBB8 9472		216	CALL DECTST			
BBBA C675		217	JZ WATCHD: LOOP IF DONE			
BBBC F1		218	MOV A,0			
BBBD 3491		219	CALL FXPRNT			
BBDF B4D2		220	JMP CRFND			
		221	REJECT			

LDC OBJ	SEQ	SOURCE STATEMENT
0100	222	ORG 100H
0100 09	223	
0101 021C	224	LDBUF: IN A,P1 ;READ PORT 1
0103 1207	225	JBS LMODE ;BIT 5 - M = LINE MODE
0105 0901	226	JBS ARND ;JUMP AROUND IF MOTOR IS ON
0107 920F	227	ORL P1,0B1H ;TURN THE MOTOR OFF
0109 FE	228	ARND: JBS MOFF ;NO FORM FEED
010A 4300	229	MOV A,LINCNT ;GET THE LINE COUNTER
010C AE	230	ORL A,000H ;SET MSB
010D 23FF	231	MOV LINCNT,A ;RESTORE THE LINE COUNTER
010F 721A	232	MOV A,00FFH ;SET ACC
0111 9478	233	MOFF: JBS MOFF ;JUMP IF NO LINE FEED
0113 09	234	CALL LINEFD ;GO DO A LF OR FF
0114 721A	235	BUTLOP: IN A,P1 ;READ THE PORT
0116 921A	236	JBS MOFF ;WAIT FOR SWITCH TO BE RELEASED
0118 2413	237	JBS MOFF ;WAIT FOR SWITCH TO BE RELEASED
011A 2400	238	JMP BUTLOP ;LOOP
	239	MOFF: JMP LDBUF ;LOOP
	240	
	241	;FIRST SEE IF A CHARACTER IS PRESENT IN THE BUFFER
	242	
011C 261F	243	LHMODE: JNB CHAR ;IF CHARACTER PRESENT, READ IT
011E 03	244	RET ;IF NOT, EXIT ROUTINE
	245	
	246	;IF THERE IS A CHARACTER, READ IT
	247	
011F FD	248	MOV A,STATUS ;GET THE STATUS
0120 5249	249	JBS ARNDJP ;IF CONTINUE IS SET, DON'T LOAD
0122 9249	250	JBS ARNDJP ;IF LF IS SET, DON'T LOAD
0124 724A	251	JBS LFCRCK ;WAS CR SET, SEE IF NEXT CHAR IS LF
0126 94D6	252	CALL GTCAR ;GO READ A CHARACTER
0128 3461	253	GOOD: CALL FXCHAR ;MAKE SURE IT IS OK
012A AB	254	MOV R1,INBUF,A ;SAVE CHARACTER IN BUFFER MEMORY
012B FD	255	MOV A,STATUS ;GET THE STATUS
012C F239	256	JBS SUB1 ;IF BIT 7 IS SET DECREMENT BUFFER
012E 18	257	INC INBUF ;UPDATE INBUF
012F 237B	258	MOV A,0MAX+1 ;GET TOP
0131 D8	259	XRL A,INBUF ;ARE WE AT THE TOP?
0132 9649	260	JNZ ARNDJP ;IF NOT GET THE STATUS
0134 F8	261	MOV A,INBUF ;GET INBUF
0135 07	262	DEC A ;CHANGE BY ONE
0136 AB	263	MOV INBUF,A ;PUT IT BACK
0137 2449	264	JMP ARNDJP ;GET THE STATUS
0139 F8	265	SUB1: MOV A,INBUF ;GET INBUF
013A 07	266	DEC A ;CHANGE BY ONE
013B AB	267	MOV INBUF,A ;PUT INBUF BACK
013C 231F	268	MOV A,0FIRST-1 ;GET THE BOTTOM OF THE BUFFER
013E D8	269	XRL A,INBUF ;TEST THE BUFFER
013F 9649	270	JNZ ARNDJP ;IF NOT ZERO READ THE STATUS
0141 18	271	INC INBUF ;MOVE INBUF BACK
0142 2449	272	JMP ARNDJP ;GO GET STATUS
0144 FD	273	GETSTA: MOV A,STATUS ;GET THE STATUS
0145 1249	274	JBS ARNDJP ;IF BIT 0 SET, BYPASS
0147 925B	275	JBS ARNDJP ;IF LF IS FOUND, SET THE STATUS
0149 03	276	ARNDJP: RET ;EXIT
	277	
	278	;THIS ROUTINE "FORCES" A LF AFTER A CR
	279	
014A 94D6	280	LFCRCK: CALL GTCAR ;READ A CHARACTER
014C 230A	281	MOV A,0BAH ;GET A LINE FEED
014E 242B	282	JMP GOOD ;JUMP BACK
	283	
	284	;THIS ROUTINE SETS THE STATUS BITS
	285	
0150 FD	286	STBIT1: MOV A,STATUS ;LOAD THE STATUS
0151 3259	287	JBS STPRNT ;IF STILL PRINTING, LEAVE
0153 4302	288	ORL A,002H ;SET PRINT BIT
0155 034B	289	ADD A,040H ;UPDATE POSITION COUNTER
0157 AD	290	MOV STATUS,A ;PUT STATUS BACK
0158 03	291	RET ;EXIT ROUTINE
0159 526B	292	STPRNT: JBS BYEBYE ;CHECK CONTINUE BIT
015B 430A	293	ORL A,004H ;SET CONTINUE BIT
015D 034B	294	ADD A,040H ;UPDATE PRINT DIRECTION
015F AD	295	MOV STATUS,A ;PUT THE STATUS BACK
0160 03	296	BYEBYE: RET ;EXIT
	297	

LOC	OBJ	SEQ	SOURCE STATEMENT	THESE STATEMENTS	LOC	OBJ
		290	:THIS ROUTINE "CONVERTS" LOWER CASE LETTERS TO			
		299	:UPPER CASE			
		300				
B161	97	301	FXCHAR: CLR C	:CLEAR THE CARRY		
B162	537F	302	ANL A, #07FH	:STRIP MSB		
B164	AF	303	MOV JUNK1, A	:SAVE ACC		
B165	B3AB	304	ADD A, #0ABH	:SEE IF NUMBER IS 6BH		
B167	E67B	305	JNC FINE	:IF CARRY ISN'T SET, JUMP		
B169	FF	306	MOV A, JUNK1	:GET ACC BACK		
B16A	37	307	CPL A	:SUBTRACT 2BH FROM THE ACC		
B16B	B32B	308	ADD A, #2BH			
B16D	37	309	CPL A			
B16E	2474	310	JMP FIXDUN	:JUMP TO TEST CR LF		
B178	37	311	FINE: CPL A	:NOW SUBTRACT ABH FROM ACC		
B171	B3AB	312	ADD A, #0ABH			
B173	37	313	CPL A			
B174	AF	314	FIXDUN: MOV JUNK1, A	:SAVE A		
B175	D3BD	315	XRL A, #0BDH	:IS CHARACTER A CR		
B177	967F	316	JNZ LFTEST	:IF IT IS NOT TEST LF		
B179	FD	317	MOV A, STATUS	:GET THE STATUS		
B17A	43B8	318	ORL A, #0BH	:SET BIT 3		
B17C	AD	319	MOV STATUS, A	:RESTORE THE STATUS		
B17D	248F	320	JMP FIXFIN	:LEAVE		
B17F	FF	321	LFTEST: MOV A, JUNK1	:GET CHARACTER BACK		
B180	D3BA	322	XRL A, #0BAH	:IS IT A LF		
B182	C689	323	JZ FIXUP	:IF ITS NOT, WE ARE DONE		
B184	FF	324	MOV A, JUNK1	:GET THE CHARACTER BACK		
B185	D3BC	325	XRL A, #0BCH	:IS IT A FORM FEED		
B187	968F	326	JNZ FIXFIN	:IF NOT FORM FEED, JUMP		
B189	FD	327	FIXUP: MOV A, STATUS	:GET THE STATUS		
B18A	431B	328	ORL A, #0BH	:SET BIT 4		
B18C	AD	329	MOV STATUS, A	:RETURN THE STATUS		
B18D	345B	330	CALL STBIT1	:SET THE STATUS		
B18F	FF	331	FIXFIN: MOV A, JUNK1	:GET THE CHARACTER		
LOC	OBJ	SEQ	SOURCE STATEMENT			
B198	83	332	RET NOT T30	:EXIT FIXCHAR		
		333				
		334	:THIS ROUTINE RECOGNIZES A LF, FF, AND CR			
		335	:DURING THE PRINT OPERATION			
		336	:IT ALSO FORCES A SPACE IF A CHARACTER FOUND			
		337	:IN THE BUFFER IS NOT IN THE LOOKUP TABLE			
		338				
B191	AF	339	FXPRNT: MOV A, JUNK1, A	:SAVE ACC		
B192	D3BC	340	XRL A, #0BCH	:FORM FEED		
B194	C6B2	341	JZ JZ TFFIX	:GO SET FORM FEED		
B196	FF	342	MOV A, JUNK1	:RESTORE CHARACTER		
B197	D3BD	343	XRL A, #0BDH	:SEE IF IT IS A CR		
B199	C6AB	344	JZ CRFIX	:LEAVE IF IT IS		
B19B	FF	345	MOV A, JUNK1	:GET ACC BACK		
B19C	D3BA	346	XRL A, #0BAH	:SEE IF IT IS A LF		
B19E	C6AB	347	JZ TFFIX	:LEAVE IF IT IS		
B1AB	FF	348	MOV A, JUNK1	:GET CHARACTER BACK		
B1A1	53EB	349	ANL A, #0EBH	:SEE IF IT IS A CHARACTER		
B1A3	96BD	350	JNZ IISCHAR	:IF IT IS JUMPL		
B1A5	232B	351	MOV A, #2BH	:PUT A SPACE IN ACC		
B1A7	83	352	RET	:EXIT		
B1AB	43B8	353	CRFIX: ORL A, #0BH	:SET BIT 7		
B1AA	83	354	RET	:EXIT		
B1AB	FD	355	FFFIX: MOV A, STATUS	:GET THE STATUS		
B1AC	432B	356	ORL A, #2BH	:SET LF BIT IN STATUS		
B1AE	AD	357	MOV STATUS, A	:PUT THE STATUS BACK		
B1AF	232B	358	MOV A, #2BH	:GET A SPACE		
B1B1	83	359	RET	:EXIT		
B1B2	FD	360	FFFIX: MOV A, STATUS	:GET THE STATUS		
B1B3	432B	361	ORL A, #2BH	:SET LINE FEED BIT		
B1B5	AD	362	MOV STATUS, A	:PUT THE STATUS BACK		
B1B6	FE	363	MOV A, LINCNT	:GET THE LINE COUNT		
B1B7	43B8	364	ORL A, #0BH	:SET BIT 7		
B1B9	AE	365	MOV A, LINCNT, A	:PUT LINE COUNT BACK		
B1BA	232B	366	MOV A, #2BH	:GET A SPACE		
B1BC	83	367	RET	:EXIT		
B1BD	FF	368	ISCHAR: MOV A, JUNK1	:GET CHARACTER BACK		
B1BE	533F	369	ANL A, #03FH	:STRIP THE TWO MSB		
B1C0	83	370	RET	:EXIT		

LOC	OBJ	SEQ	SOURCE STATEMENT	THRESTATE	33RU02	032	400	504
		371						
		372	THIS ROUTINE PRINTS THE CHARACTER IN THE ACC					
		373						
B1C1	AC	374	PRNTIT: MOV TEMP1,A					
B1C2	E7	375	RL A					
B1C3	E7	376	RL A					
B1C4	6C	377	ADD A,TEMP1					
		378						
		379	NOW SEE WHAT PART OF THE LOOKUP TABLE TO USE					
		380						
B1C5	2C	381	XCH A,TEMP1					
B1C6	B2CA	382	JB5 SHORT					
B1C8	44AB	383	JMP PAGE1					
B1CA	64AB	384	SHORT: JMP PAGE2					
		385						
		386	THIS ROUTINE TRIGGERS THE SOLENOIDS FOR 600 MICROSECONDS					
		387	AFTER WAITING FOR THE TRIGGER SIGNAL FROM THE PRINTER					
		388						
B1CC	AF	389	FIRE: MOV JUNK1,A					
B1CD	FD	390	MOV A,STATUS					
B1CE	D2D4	391	JB6 NT1					
B1DB	56DB	392	FIREX: JTI FIREX					
B1DE	24D6	393	JMP FIREY					
B1D4	46D4	394	NT1: JMT1 NT1					
B1D6	FF	395	FIREY: MOV A,JUNK1					
B1D7	9B	396	MOVX BRB,A					
		397						
		398	NOW KILL 600 MICROSECONDS					
		399						
B1DB	23F3	400	MOV A,#BF3H					
B1DA	62	401	MOV T,A					
B1DB	55	402	STRT T					
B1DC	16E0	403	TSJTF: JTF KTDUN					
B1DE	24DC	404	JMP TSJTF					
B1EB	27	405	KTDUN: CLR A					
B1E1	9B	406	MOVX BRB,A					
B1E2	65	407	STOP TCNT					
B1E3	03	408	RET					
		409	SEJECT					

LOC	OBJ	SEQ	SOURCE STATEMENT	THIRTY STATE	LOC	JOB
		410	;			
		411	;			
		412	;			
		413	THIS IS THE LOOKUP TABLE. THE MSB IS NOT USED. THE MSB - 1			
		414	IS THE DOT THAT IS THE TOP OF ANY GIVEN CHARACTER AND THE			
		415	LSB IS THE DOT THAT IS THE BOTTOM OF ANY GIVEN CHARACTER			
		416	;			
		417	;			
		418	;			
0200		419	ORG 200H			
		420	;			
0200	3E	421	TABLE1: DB 3EH			
0201	41	422	DB 41H			
0202	5D	423	DB 5DH			
0203	59	424	DB 59H			
0204	4E	425	DB 4EH			
		426	;			
0205	7C	427	DB 7CH			
0206	12	428	DB 12H			
0207	11	429	DB 11H			
0208	12	430	DB 12H			
0209	7C	431	DB 7CH			
		432	;			
020A	7F	433	DB 7FH			
020B	49	434	DB 49H			
020C	49	435	DB 49H			
020D	49	436	DB 49H			
020E	36	437	DB 36H			
		438	;			
020F	3E	439	DB 3EH			
0210	41	440	DB 41H			
0211	41	441	DB 41H			
0212	41	442	DB 41H			
0213	22	443	DB 22H			
		444	;			
0214	7F	445	DB 7FH			
0215	41	446	DB 41H			
0216	41	447	DB 41H			
0217	41	448	DB 41H			
0218	3E	449	DB 3EH			
		450	;			
0219	7F	451	DB 7FH			
021A	49	452	DB 49H			
021B	49	453	DB 49H			
021C	49	454	DB 49H			
021D	41	455	DB 41H			
		456	*EJECT			

LOC	OBJ	SEQ	SOURCE STATEMENT	THANETATB 309402	Q30	LOC	OBJ
		457			010		
021E	7F	458	DB ***** 7FH	: H7F ***** 00	010	00	010
021F	09	459	DB ***** 09H	: H10 ***** 00	010	00	010
0220	09	460	DB ***** 09H	: H10 ***** 00	010	00	010
0221	09	461	DB ***** 09H	: H10 ***** 00	010	00	010
0222	01	462	DB ***** 01H	: H01 ***** 00	010	00	010
		463			010		
0223	3E	464	DB ***** 3EH	: H3E ***** 00	010	00	010
0224	41	465	DB ***** 41H	: H41 ***** 00	010	00	010
0225	41	466	DB ***** 41H	: H41 ***** 00	010	00	010
0226	51	467	DB ***** 51H	: H51 ***** 00	010	00	010
0227	71	468	DB ***** 71H	: H71 ***** 00	010	00	010
		469			010		
0228	7F	470	DB ***** 7FH	: H7F ***** 00	010	00	010
0229	08	471	DB ***** 08H	: H08 ***** 00	010	00	010
022A	08	472	DB ***** 08H	: H08 ***** 00	010	00	010
022B	08	473	DB ***** 08H	: H08 ***** 00	010	00	010
022C	7F	474	DB ***** 7FH	: H7F ***** 00	010	00	010
		475			010		
022D	08	476	DB ***** 08H	: H08 ***** 00	010	00	010
022E	41	477	DB ***** 41H	: H41 ***** 00	010	00	010
022F	7F	478	DB ***** 7FH	: H7F ***** 00	010	00	010
0230	41	479	DB ***** 41H	: H41 ***** 00	010	00	010
0231	08	480	DB ***** 08H	: H08 ***** 00	010	00	010
		481			010		
0232	20	482	DB ***** 20H	: H20 ***** 00	010	00	010
0233	40	483	DB ***** 40H	: H40 ***** 00	010	00	010
0234	40	484	DB ***** 40H	: H40 ***** 00	010	00	010
0235	40	485	DB ***** 40H	: H40 ***** 00	010	00	010
0236	3F	486	DB ***** 3FH	: H3F ***** 00	010	00	010
		487			010		
0237	7F	488	DB ***** 7FH	: H7F ***** 00	010	00	010
0238	08	489	DB ***** 08H	: H08 ***** 00	010	00	010
0239	14	490	DB ***** 14H	: H14 ***** 00	010	00	010
023A	22	491	DB ***** 22H	: H22 ***** 00	010	00	010
023B	41	492	DB ***** 41H	: H41 ***** 00	010	00	010
		493			010		
023C	7F	494	DB ***** 7FH	: H7F ***** 00	010	00	010
023D	40	495	DB ***** 40H	: H40 ***** 00	010	00	010
023E	40	496	DB ***** 40H	: H40 ***** 00	010	00	010
023F	40	497	DB ***** 40H	: H40 ***** 00	010	00	010
0240	40	498	DB ***** 40H	: H40 ***** 00	010	00	010
		499			010		
0241	7F	500	DB ***** 7FH	: H7F ***** 00	010	00	010
0242	02	501	DB ***** 02H	: H02 ***** 00	010	00	010
0243	0C	502	DB ***** 0CH	: H0C ***** 00	010	00	010
0244	02	503	DB ***** 02H	: H02 ***** 00	010	00	010
0245	7F	504	DB ***** 7FH	: H7F ***** 00	010	00	010
		505			010		
0246	7F	506	DB ***** 7FH	: H7F ***** 00	010	00	010
0247	04	507	DB ***** 04H	: H04 ***** 00	010	00	010
0248	08	508	DB ***** 08H	: H08 ***** 00	010	00	010
0249	10	509	DB ***** 10H	: H10 ***** 00	010	00	010
024A	7F	510	DB ***** 7FH	: H7F ***** 00	010	00	010
		511	*EJECT		010		

T0343 002

LDC	OBJ	SEQ	SOURCE STATEMENT	INHERITATE STATEMENT	OBJ	SEQ
		512				
0248	3E	513	DB 3EH	DB 3EH	0248	3E
024C	41	514	DB 41H	DB 41H	024C	41
024D	41	515	DB 41H	DB 41H	024D	41
024E	41	516	DB 41H	DB 41H	024E	41
024F	3E	517	DB 3EH	DB 3EH	024F	3E
		518				
0250	7F	519	DB 7FH	DB 7FH	0250	7F
0251	09	520	DB 09H	DB 09H	0251	09
0252	09	521	DB 09H	DB 09H	0252	09
0253	09	522	DB 09H	DB 09H	0253	09
0254	06	523	DB 06H	DB 06H	0254	06
		524				
0255	3E	525	DB 3EH	DB 3EH	0255	3E
0256	41	526	DB 41H	DB 41H	0256	41
0257	51	527	DB 51H	DB 51H	0257	51
0258	21	528	DB 21H	DB 21H	0258	21
0259	5E	529	DB 5EH	DB 5EH	0259	5E
		530				
025A	7F	531	DB 7FH	DB 7FH	025A	7F
025B	09	532	DB 09H	DB 09H	025B	09
025C	19	533	DB 19H	DB 19H	025C	19
025D	29	534	DB 29H	DB 29H	025D	29
025E	46	535	DB 46H	DB 46H	025E	46
		536				
025F	26	537	DB 26H	DB 26H	025F	26
0260	49	538	DB 49H	DB 49H	0260	49
0261	49	539	DB 49H	DB 49H	0261	49
0262	49	540	DB 49H	DB 49H	0262	49
0263	32	541	DB 32H	DB 32H	0263	32
		542				
0264	01	543	DB 01H	DB 01H	0264	01
0265	01	544	DB 01H	DB 01H	0265	01
0266	7F	545	DB 7FH	DB 7FH	0266	7F
0267	01	546	DB 01H	DB 01H	0267	01
0268	01	547	DB 01H	DB 01H	0268	01
		548				
0269	3F	549	DB 3FH	DB 3FH	0269	3F
026A	40	550	DB 40H	DB 40H	026A	40
026B	40	551	DB 40H	DB 40H	026B	40
026C	40	552	DB 40H	DB 40H	026C	40
026D	3F	553	DB 3FH	DB 3FH	026D	3F
		554				
026E	1F	555	DB 1FH	DB 1FH	026E	1F
026F	20	556	DB 20H	DB 20H	026F	20
0270	40	557	DB 40H	DB 40H	0270	40
0271	20	558	DB 20H	DB 20H	0271	20
0272	1F	559	DB 1FH	DB 1FH	0272	1F
		560				
0273	7F	561	DB 7FH	DB 7FH	0273	7F
0274	20	562	DB 20H	DB 20H	0274	20
0275	10	563	DB 10H	DB 10H	0275	10
0276	20	564	DB 20H	DB 20H	0276	20
0277	7F	565	DB 7FH	DB 7FH	0277	7F
		566	*EJECT			

LOC	OBJ	SEQ	SOURCE STATEMENT	THEN STATE	339002	01	LBO	100
0278	63	567	DB 0278 163H	DB 0278 163H	DB 0278 163H	DB 0278 163H	DB 0278 163H	DB 0278 163H
0279	14	569	DB 0279 14H	DB 0279 14H	DB 0279 14H	DB 0279 14H	DB 0279 14H	DB 0279 14H
027A	08	570	DB 027A 08H	DB 027A 08H	DB 027A 08H	DB 027A 08H	DB 027A 08H	DB 027A 08H
027B	14	571	DB 027B 14H	DB 027B 14H	DB 027B 14H	DB 027B 14H	DB 027B 14H	DB 027B 14H
027C	63	572	DB 027C 63H	DB 027C 63H	DB 027C 63H	DB 027C 63H	DB 027C 63H	DB 027C 63H
027D	03	573	DB 027D 03H	DB 027D 03H	DB 027D 03H	DB 027D 03H	DB 027D 03H	DB 027D 03H
027E	04	574	DB 027E 04H	DB 027E 04H	DB 027E 04H	DB 027E 04H	DB 027E 04H	DB 027E 04H
027F	78	575	DB 027F 78H	DB 027F 78H	DB 027F 78H	DB 027F 78H	DB 027F 78H	DB 027F 78H
0280	04	576	DB 0280 04H	DB 0280 04H	DB 0280 04H	DB 0280 04H	DB 0280 04H	DB 0280 04H
0281	03	577	DB 0281 03H	DB 0281 03H	DB 0281 03H	DB 0281 03H	DB 0281 03H	DB 0281 03H
0282	61	578	DB 0282 61H	DB 0282 61H	DB 0282 61H	DB 0282 61H	DB 0282 61H	DB 0282 61H
0283	51	579	DB 0283 51H	DB 0283 51H	DB 0283 51H	DB 0283 51H	DB 0283 51H	DB 0283 51H
0284	49	580	DB 0284 49H	DB 0284 49H	DB 0284 49H	DB 0284 49H	DB 0284 49H	DB 0284 49H
0285	45	581	DB 0285 45H	DB 0285 45H	DB 0285 45H	DB 0285 45H	DB 0285 45H	DB 0285 45H
0286	43	582	DB 0286 43H	DB 0286 43H	DB 0286 43H	DB 0286 43H	DB 0286 43H	DB 0286 43H
0287	7F	583	DB 0287 7FH	DB 0287 7FH	DB 0287 7FH	DB 0287 7FH	DB 0287 7FH	DB 0287 7FH
0288	7F	584	DB 0288 7FH	DB 0288 7FH	DB 0288 7FH	DB 0288 7FH	DB 0288 7FH	DB 0288 7FH
0289	41	585	DB 0289 41H	DB 0289 41H	DB 0289 41H	DB 0289 41H	DB 0289 41H	DB 0289 41H
028A	41	586	DB 028A 41H	DB 028A 41H	DB 028A 41H	DB 028A 41H	DB 028A 41H	DB 028A 41H
028B	41	587	DB 028B 41H	DB 028B 41H	DB 028B 41H	DB 028B 41H	DB 028B 41H	DB 028B 41H
028C	02	588	DB 028C 02H	DB 028C 02H	DB 028C 02H	DB 028C 02H	DB 028C 02H	DB 028C 02H
028D	04	589	DB 028D 04H	DB 028D 04H	DB 028D 04H	DB 028D 04H	DB 028D 04H	DB 028D 04H
028E	08	590	DB 028E 08H	DB 028E 08H	DB 028E 08H	DB 028E 08H	DB 028E 08H	DB 028E 08H
028F	10	591	DB 028F 10H	DB 028F 10H	DB 028F 10H	DB 028F 10H	DB 028F 10H	DB 028F 10H
0290	20	592	DB 0290 20H	DB 0290 20H	DB 0290 20H	DB 0290 20H	DB 0290 20H	DB 0290 20H
0291	41	593	DB 0291 41H	DB 0291 41H	DB 0291 41H	DB 0291 41H	DB 0291 41H	DB 0291 41H
0292	41	594	DB 0292 41H	DB 0292 41H	DB 0292 41H	DB 0292 41H	DB 0292 41H	DB 0292 41H
0293	41	595	DB 0293 41H	DB 0293 41H	DB 0293 41H	DB 0293 41H	DB 0293 41H	DB 0293 41H
0294	7F	596	DB 0294 7FH	DB 0294 7FH	DB 0294 7FH	DB 0294 7FH	DB 0294 7FH	DB 0294 7FH
0295	7F	597	DB 0295 7FH	DB 0295 7FH	DB 0295 7FH	DB 0295 7FH	DB 0295 7FH	DB 0295 7FH
0296	10	598	DB 0296 10H	DB 0296 10H	DB 0296 10H	DB 0296 10H	DB 0296 10H	DB 0296 10H
0297	08	599	DB 0297 08H	DB 0297 08H	DB 0297 08H	DB 0297 08H	DB 0297 08H	DB 0297 08H
0298	04	600	DB 0298 04H	DB 0298 04H	DB 0298 04H	DB 0298 04H	DB 0298 04H	DB 0298 04H
0299	08	601	DB 0299 08H	DB 0299 08H	DB 0299 08H	DB 0299 08H	DB 0299 08H	DB 0299 08H
029A	10	602	DB 029A 10H	DB 029A 10H	DB 029A 10H	DB 029A 10H	DB 029A 10H	DB 029A 10H
029B	40	603	DB 029B 40H	DB 029B 40H	DB 029B 40H	DB 029B 40H	DB 029B 40H	DB 029B 40H
029C	40	604	DB 029C 40H	DB 029C 40H	DB 029C 40H	DB 029C 40H	DB 029C 40H	DB 029C 40H
029D	40	605	DB 029D 40H	DB 029D 40H	DB 029D 40H	DB 029D 40H	DB 029D 40H	DB 029D 40H
029E	40	606	DB 029E 40H	DB 029E 40H	DB 029E 40H	DB 029E 40H	DB 029E 40H	DB 029E 40H
029F	40	607	DB 029F 40H	DB 029F 40H	DB 029F 40H	DB 029F 40H	DB 029F 40H	DB 029F 40H
		608	DB 029F 40H	DB 029F 40H	DB 029F 40H	DB 029F 40H	DB 029F 40H	DB 029F 40H
		609	DB 029F 40H	DB 029F 40H	DB 029F 40H	DB 029F 40H	DB 029F 40H	DB 029F 40H
		610	DB 029F 40H	DB 029F 40H	DB 029F 40H	DB 029F 40H	DB 029F 40H	DB 029F 40H
		611	DB 029F 40H	DB 029F 40H	DB 029F 40H	DB 029F 40H	DB 029F 40H	DB 029F 40H
		612	DB 029F 40H	DB 029F 40H	DB 029F 40H	DB 029F 40H	DB 029F 40H	DB 029F 40H
		613	DB 029F 40H	DB 029F 40H	DB 029F 40H	DB 029F 40H	DB 029F 40H	DB 029F 40H
		614	DB 029F 40H	DB 029F 40H	DB 029F 40H	DB 029F 40H	DB 029F 40H	DB 029F 40H
		615	DB 029F 40H	DB 029F 40H	DB 029F 40H	DB 029F 40H	DB 029F 40H	DB 029F 40H

LDC	OBJ	SEQ	SOURCE STATEMENT	THRESTATE 335002	NS	LAB	301
		616	;		730		
B2A8 B8B8		617 PAGE1:	MOV STBCNT, #00H	;ZERO STROBE COUNTER	840	00	8150
B2A2 FA		618	MOV A, SAVPNT	;GET DIRECTION	850	01	8150
B2A3 37		619	CPL A	;FLIP BITS	860	00	8150
B2A4 D2B3		620	JB6 BAKWRD	;IF BACKWARD JUMP OUT	870	01	8150
B2A6 FC		621 LKLO:	MOV A, TEMP1	;GET THE TARGET	880	00	8150
B2A7 A3		622	MOV A, #A	;GET THE DATA	890	00	8150
B2A8 34CC		623	CALL FIRE	;STROBE THE SOLENOIDS	900	00	8150
B2AA 1C		624	INC TEMP1	;INCREMENT THE POINTER	910	00	8150
B2AB 1B		625	INC STBCNT	;INCREMENT THE STROBE COUNTER	920	00	8150
B2AC FB		626	MOV A, STBCNT	;GET THE STROBE COUNTER	930	00	8150
B2AD D3B5		627	XRL A, #B5H	;IS IT FIVE	940	00	8150
B2AF 96A6		628	JNZ LKLO	;REPEAT IF NOT FIVE	950	00	8150
B2B1 84AE		629	JMP SETTIM	;GO BACK	960	00	8150
B2B3 FC		630 BAKWRD:	MOV A, TEMP1	;GET THE TARGET	970	00	8150
B2B4 B3B4		631	ADD A, #B4H	;COMPENSATE FOR GOING BACKWARDS	980	00	8150
B2B6 AC		632	MOV TEMP1, A	;SAVE IT	990	00	8150
B2B7 FC		633 LKLO1:	MOV A, TEMP1	;GET THE TARGET	1000	00	8150
B2B8 A3		634	MOV A, #A	;GET THE DATA	1010	00	8150
B2B9 34CC		635	CALL FIRE	;STROBE THE SOLENOIDS	1020	00	8150
B2BB FC		636	MOV A, TEMP1	;GET TEMP1	1030	00	8150
B2BC B7		637	DEC A	;DECREASE BY ONE	1040	00	8150
B2BD AC		638	MOV TEMP1, A	;PUT IT BACK	1050	00	8150
B2BE 1B		639	INC STBCNT	;INCREMENT THE STROBE COUNTER	1060	00	8150
B2BF FB		640	MOV A, STBCNT	;GET THE STROBE COUNTER	1070	00	8150
B2CB D3B5		641	XRL A, #B5H	;IS IT FIVE	1080	00	8150
B2C2 96B7		642	JNZ LKLO1	;REPEAT IF NOT FIVE	1090	00	8150
B2C4 84AE		643	JMP SETTIM	;GO BACK, CHARACTER IS DONE	1100	00	8150
		644 \$EJECT			1110	00	8150
					1120	00	8150
					1130	00	8150
					1140	00	8150
					1150	00	8150
					1160	00	8150
					1170	00	8150
					1180	00	8150
					1190	00	8150
					1200	00	8150
					1210	00	8150
					1220	00	8150
					1230	00	8150
					1240	00	8150
					1250	00	8150
					1260	00	8150
					1270	00	8150
					1280	00	8150
					1290	00	8150
					1300	00	8150
					1310	00	8150
					1320	00	8150
					1330	00	8150
					1340	00	8150
					1350	00	8150
					1360	00	8150
					1370	00	8150
					1380	00	8150
					1390	00	8150
					1400	00	8150
					1410	00	8150
					1420	00	8150
					1430	00	8150
					1440	00	8150
					1450	00	8150
					1460	00	8150
					1470	00	8150
					1480	00	8150
					1490	00	8150
					1500	00	8150
					1510	00	8150
					1520	00	8150
					1530	00	8150
					1540	00	8150
					1550	00	8150
					1560	00	8150
					1570	00	8150
					1580	00	8150
					1590	00	8150
					1600	00	8150
					1610	00	8150
					1620	00	8150
					1630	00	8150
					1640	00	8150
					1650	00	8150
					1660	00	8150
					1670	00	8150
					1680	00	8150
					1690	00	8150
					1700	00	8150
					1710	00	8150
					1720	00	8150
					1730	00	8150
					1740	00	8150
					1750	00	8150
					1760	00	8150
					1770	00	8150
					1780	00	8150
					1790	00	8150
					1800	00	8150
					1810	00	8150
					1820	00	8150
					1830	00	8150
					1840	00	8150
					1850	00	8150
					1860	00	8150
					1870	00	8150
					1880	00	8150
					1890	00	8150
					1900	00	8150
					1910	00	8150
					1920	00	8150
					1930	00	8150
					1940	00	8150
					1950	00	8150
					1960	00	8150
					1970	00	8150
					1980	00	8150
					1990	00	8150
					2000	00	8150

LUC	OBJ	DEM	SOURCE	STATION	THRESHOLD	32000	932	640
		645	1*					
0300		646	ORG	300H				
		647	1*					
		648						
0300	00	649	DB	00H				
0301	00	650	DB	00H				
0302	00	651	DB	00H				
0303	00	652	DB	00H				
0304	00	653	DB	00H				
		654						
0305	00	655	DB	00H				
0306	00	656	DB	00H				
0307	5F	657	DB	5FH				
0308	00	658	DB	00H				
0309	00	659	DB	00H				
		660						
030A	00	661	DB	00H				
030B	07	662	DB	07H				
030C	00	663	DB	00H				
030D	07	664	DB	07H				
030E	00	665	DB	00H				
		666						
030F	14	667	DB	14H				
0310	7F	668	DB	7FH				
0311	14	669	DB	14H				
0312	7F	670	DB	7FH				
0313	14	671	DB	14H				
		672						
0314	24	673	DB	24H				
0315	2A	674	DB	2AH				
0316	7F	675	DB	7FH				
0317	2A	676	DB	2AH				
0318	12	677	DB	12H				
		678						
0319	23	679	DB	23H				
031A	13	680	DB	13H				
031B	00	681	DB	00H				
031C	64	682	DB	64H				
031D	62	683	DB	62H				
		684						
031E	36	685	DB	36H				
031F	49	686	DB	49H				
0320	56	687	DB	56H				
0321	20	688	DB	20H				
0322	50	689	DB	50H				
		690	*EJECT					

LDC	OBJ	SEQ	SOURCE STATEMENT	THRESTATE SOURCE	SEQ	OBJ
		691			691	
0323	00	692	DB 00H	DB 00H	692	00H
0324	00	693	DB 00H	DB 00H	693	00H
0325	07	694	DB 07H	DB 07H	694	07H
0326	00	695	DB 00H	DB 00H	695	00H
0327	00	696	DB 00H	DB 00H	696	00H
		697			697	
0328	1C	698	DB 1CH	DB 1CH	698	1CH
0329	22	699	DB 22H	DB 22H	699	22H
032A	41	700	DB 41H	DB 41H	700	41H
032B	00	701	DB 00H	DB 00H	701	00H
032C	00	702	DB 00H	DB 00H	702	00H
		703			703	
032D	00	704	DB 00H	DB 00H	704	00H
032E	00	705	DB 00H	DB 00H	705	00H
032F	41	706	DB 41H	DB 41H	706	41H
0330	22	707	DB 22H	DB 22H	707	22H
0331	1C	708	DB 1CH	DB 1CH	708	1CH
		709			709	
0332	22	710	DB 22H	DB 22H	710	22H
0333	14	711	DB 14H	DB 14H	711	14H
0334	7F	712	DB 7FH	DB 7FH	712	7FH
0335	14	713	DB 14H	DB 14H	713	14H
0336	22	714	DB 22H	DB 22H	714	22H
		715			715	
0337	00	716	DB 00H	DB 00H	716	00H
0338	00	717	DB 00H	DB 00H	717	00H
0339	7F	718	DB 7FH	DB 7FH	718	7FH
033A	00	719	DB 00H	DB 00H	719	00H
033B	00	720	DB 00H	DB 00H	720	00H
		721			721	
033C	00	722	DB 00H	DB 00H	722	00H
033D	40	723	DB 40H	DB 40H	723	40H
033E	30	724	DB 30H	DB 30H	724	30H
033F	00	725	DB 00H	DB 00H	725	00H
0340	00	726	DB 00H	DB 00H	726	00H
		727			727	
0341	00	728	DB 00H	DB 00H	728	00H
0342	00	729	DB 00H	DB 00H	729	00H
0343	00	730	DB 00H	DB 00H	730	00H
0344	00	731	DB 00H	DB 00H	731	00H
0345	00	732	DB 00H	DB 00H	732	00H
		733			733	
0346	00	734	DB 00H	DB 00H	734	00H
0347	00	735	DB 00H	DB 00H	735	00H
0348	40	736	DB 40H	DB 40H	736	40H
0349	00	737	DB 00H	DB 00H	737	00H
034A	00	738	DB 00H	DB 00H	738	00H
		739			739	
034B	20	740	DB 20H	DB 20H	740	20H
034C	10	741	DB 10H	DB 10H	741	10H
034D	00	742	DB 00H	DB 00H	742	00H
034E	04	743	DB 04H	DB 04H	743	04H
034F	02	744	DB 02H	DB 02H	744	02H
		745			745	
0350	3E	746	DB 3EH	DB 3EH	746	3EH
0351	51	747	DB 51H	DB 51H	747	51H
0352	49	748	DB 49H	DB 49H	748	49H
0353	45	749	DB 45H	DB 45H	749	45H
0354	3E	750	DB 3EH	DB 3EH	750	3EH
		751			751	
0355	00	752	DB 00H	DB 00H	752	00H
0356	42	753	DB 42H	DB 42H	753	42H
0357	7F	754	DB 7FH	DB 7FH	754	7FH
0358	40	755	DB 40H	DB 40H	755	40H
0359	00	756	DB 00H	DB 00H	756	00H
		757			757	
035A	62	758	DB 62H	DB 62H	758	62H
035B	51	759	DB 51H	DB 51H	759	51H
035C	49	760	DB 49H	DB 49H	760	49H
035D	49	761	DB 49H	DB 49H	761	49H
035E	46	762	DB 46H	DB 46H	762	46H
		763			763	
035F	21	764	DB 21H	DB 21H	764	21H
0360	41	765	DB 41H	DB 41H	765	41H

LDC	OBJ	SEQ	SOURCE STATEMENT	TRANSLATE 33RUC2	OBJ
		800			
037D	46	801	DB 46H	: *1* ** 00	00
037E	49	802	DB 49H	: *1* * 00	00
037F	49	803	DB 49H	: * * * 00	00
0380	29	804	DB 29H	: *1* * 00	00
0381	1E	805	DB 1EH	: *1* *** 00	00
		806			
0382	00	807	DB 00H	: *1* 00	00
0383	00	808	DB 00H	: *1* 00	00
0384	14	809	DB 14H	: * * 00	00
0385	00	810	DB 00H	: *1* 00	00
0386	00	811	DB 00H	: *1* 00	00
		812			
0387	00	813	DB 00H	: *1* 00	00
0388	40	814	DB 40H	: *1* 00	00
0389	34	815	DB 34H	: * * * 00	00
038A	00	816	DB 00H	: *1* 00	00
038B	00	817	DB 00H	: *1* 00	00
		818			
038C	00	819	DB 00H	: *1* * 00	00
038D	14	820	DB 14H	: *1* * 00	00
038E	22	821	DB 22H	: * * * 00	00
038F	41	822	DB 41H	: *1* * 00	00
0390	00	823	DB 00H	: *1* 00	00
		824			
0391	14	825	DB 14H	: *1* * 00	00
0392	14	826	DB 14H	: *1* * 00	00
0393	14	827	DB 14H	: * * * 00	00
0394	14	828	DB 14H	: *1* * 00	00
0395	14	829	DB 14H	: *1* * 00	00
		830			
0396	00	831	DB 00H	: *1* 00	00
0397	41	832	DB 41H	: *1* * 00	00
0398	22	833	DB 22H	: * * * 00	00
0399	14	834	DB 14H	: * * * 00	00
039A	00	835	DB 00H	: * 00	00
		836			
039B	02	837	DB 02H	: * 00	00
039C	01	838	DB 01H	: * * * 00	00
039D	59	839	DB 59H	: * * * 00	00
039E	05	840	DB 05H	: * * * 00	00
039F	02	841	DB 02H	: * 00	00
		842	#EJECT		

LOC	OBJ	SEM	SOURCE	SIN	CHG	LOC	OBJ	SEM	SOURCE	SIN	CHG
03A0	0800	843	PAGE2:	MOV	STBCNT, 000H	03A0	0800	843	PAGE2:	MOV	STBCNT, 000H
03A2	FA	844		MOV	A, SAVPNT	03A2	FA	844		MOV	A, SAVPNT
03A3	37	845		CPL	A	03A3	37	845		CPL	A
03A4	02B5	846		JB6	00A BKWRD	03A4	02B5	846		JB6	00A BKWRD
03A6	FC	847	LKH11:	MOV	A, TEMP1	03A6	FC	847	LKH11:	MOV	A, TEMP1
03A7	036B	848		ADD	A, 060H	03A7	036B	848		ADD	A, 060H
03A9	A3	849		MOV	A, 0A	03A9	A3	849		MOV	A, 0A
03AA	34CC	850		CALL	FIRE	03AA	34CC	850		CALL	FIRE
03AC	1C	851		INC	TEMP1	03AC	1C	851		INC	TEMP1
03AD	10	852		INC	STBCNT	03AD	10	852		INC	STBCNT
03AE	F8	853		MOV	A, STBCNT	03AE	F8	853		MOV	A, STBCNT
03AF	03B5	854		XRL	A, 0B5H	03AF	03B5	854		XRL	A, 0B5H
03B1	96A6	855		JNZ	LKH1	03B1	96A6	855		JNZ	LKH1
03B3	84AE	856		JMP	SETTIN	03B3	84AE	856		JMP	SETTIN
03B5	FC	857	BKWRD:	MOV	A, TEMP1	03B5	FC	857	BKWRD:	MOV	A, TEMP1
03B6	0364	858		ADD	A, 064H	03B6	0364	858		ADD	A, 064H
03B8	AC	859		MOV	TEMP1, A	03B8	AC	859		MOV	TEMP1, A
03B9	FC	860	LKH11:	MOV	A, TEMP1	03B9	FC	860	LKH11:	MOV	A, TEMP1
03BA	A3	861		MOV	A, 0A	03BA	A3	861		MOV	A, 0A
03BB	34CC	862		CALL	FIRE	03BB	34CC	862		CALL	FIRE
03BD	FC	863		MOV	A, TEMP1	03BD	FC	863		MOV	A, TEMP1
03BE	07	864		DEC	A	03BE	07	864		DEC	A
03BF	AC	865		MOV	TEMP1, A	03BF	AC	865		MOV	TEMP1, A
03C0	1B	866		INC	STBCNT	03C0	1B	866		INC	STBCNT
03C1	F8	867		MOV	A, STBCNT	03C1	F8	867		MOV	A, STBCNT
03C2	03B5	868		XRL	A, 0B5H	03C2	03B5	868		XRL	A, 0B5H
03C4	96B9	869		JNZ	LKH11	03C4	96B9	869		JNZ	LKH11
03C6	84AE	870		JMP	SETTIN	03C6	84AE	870		JMP	SETTIN
		871	\$EJECT					871	\$EJECT		

LDC OBJ	SEQ	SOURCE STATEMENT	
	872	ORG 4000H	
8400	873	ORG 4000H	
	874		
8400 27	875	CLR A	ZERO ACC
8401 90	876	MOVX PR0, A	TURN OFF THE SOLENOIDS
8402 9400	877	CALL SETUP	SET UP THE PRINTER
8404 943F	878	CALL VARSET	SET UP THE SOFTWARE
8406 840A	879	JMP PRNT	GO START
	880		
8408 23FE	881	MOV A, #0FEH	LOAD ACC WITH VALUE TO TURN ON MOTOR
840A 39	882	OUTL P1, A	TURN ON MOTOR
	883		
	884	MOV DELAY, 3.2 SECONDS	WHILE CHECKING RIGHT SENSOR
	885		
8408 BC05	886	MOV TEMP1, #05H	LOAD DELAY VALUE ONE
840D BFFF	887	SELFC: MOV JUNK1, #0FFH	LOAD DELAY VALUE TWO
840F BEFF	888	SELFB: MOV LINCNT, #0FFH	LOAD DELAY VALUE THREE
8411 09	889	SELFA: IN A, P1	READ PORT ONE
8412 37	890	CPL A	MAKE THINGS RIGHT
8413 F21D	891	JB7 DONE1	IS BIT 7 SET?
8415 EE11	892	DJNZ LINCNT, SELFA	SMALL LOOP
8417 EF0F	893	DJNZ JUNK1, SELFB	BIGGER LOOP
8419 EC0D	894	DJNZ TEMP1, SELFC	BIGGEST LOOP
841B 845A	895	JMP ERROR	SOMETHING IS WRONG
	896		
	897	MOV MAKE SURE THE RIGHT	SENSOR IS CLEARED
	898		
841D BFFF	899	DONE1: MOV JUNK1, #0FFH	SET UP DELAY
841F BEFF	900	SELF: MOV LINCNT, #0FFH	SOME MORE DELAY
8421 09	901	SELFI: IN A, P1	GET THE FLAG INFORMATION
8422 F22A	902	JB7 DONE1	IS FLAG CLEARED?
8424 EE21	903	DJNZ LINCNT, SELFI	IF NOT LOOP
8426 EF1F	904	DJNZ JUNK1, SELF	LOOP SOME MORE
8428 845A	905	JMP ERROR	LEAVE IF FLAG IS NOT UNCOVERED
	906		
	907	MOV CHECK THE LEFT SENSOR IN THE SAME MANNER AS THE	
	908	RIGHT SENSOR. EXCEPT DELAY ONLY 2.5 SECONDS	
	909		
842A BC04	910	DONE1: MOV TEMP1, #04H	LOAD DELAY 1
842C BFFF	911	SELFC: MOV JUNK1, #0FFH	LOAD DELAY 2
842E BEFF	912	SELFB: MOV LINCNT, #0FFH	LOAD DELAY 3
8430 09	913	SELFA: IN A, P1	READ THE PORT
8431 37	914	CPL A	CHANGE THINGS AROUND
8432 D23C	915	JB6 DONE1	OK IF BIT 6 IS A ZERO
8434 EE30	916	DJNZ LINCNT, SELFA	SMALL LOOP
8436 EF2E	917	DJNZ JUNK1, SELFB	BIGGER LOOP
8438 EC2C	918	DJNZ TEMP1, SELFC	BIGGEST LOOP
843A 845A	919	JMP ERROR	SOMETHING IS WRONG
843C 8901	920	DONE1: ORL P1, #01H	TURN MOTOR OFF
843E 03	921	RET	GO BACK
	922		
	923	MOV SET UP THE VARIABLES	
	924		
843F 23FE	925	VARSET: MOV A, #0FEH	LOAD THE TIMER
8441 62	926	MOV T, A	
8442 55	927	STRT T	START THE TIMER
8443 8020	928	MOV INBUF, #FIRST	LOAD INPUT BUFFER
8445 BE00	929	MOV LINCNT, #00H	SET LINE COUNT
8447 B000	930	MOV STATUS, #00H	SET FORWARD BIT
	931		
	932	MOV CLEAR THE RAM AREA BY WRITING SPACE CODES	
	933		
8449 B920	934	MOV OUTBUF, #FIRST	LOAD OUTBUF
844B 2320	935	CLRMEM: MOV A, #20H	PUT SPACE CODE IN ACC
844D A1	936	MOV @OUTBUF, A	PUT SPACE CODE IN DATA MEMORY
844E 19	937	INC OUTBUF	UPDATE THE POINTER
844F F9	938	MOV A, OUTBUF	MOVE THE POINTER INTO ACC
8450 D370	939	XRL A, #00H	SEE IF DONE
8452 9640	940	JNZ CLRMEM	LOOP IF NOT CLEARED
	941		
	942	MOV CLEAR THE 8212	
	943		
8454 99EF	944	ANL P1, #0FEH	SET ENABLE BIT
8456 00	945	MOVX A, @INBUF	CLEAR THE 8212 INPUT BUFFER
8457 0910	946	ORL P1, #01H	RESET ENABLE BIT
	947		

LDC OBJ	SEQ	SOURCE STATEMENT	THANETATS 330002	0	480
	948	:NOW EXIT VARSET			
	949	:NOW EXIT T30			
B459 83	950	:LEAVE INITIALIZATION			
	951	:			
	952	:THIS ROUTINE TURNS THE MOTOR OFF AND LOOPS			
	953	:			
B45A 89FF	954	ERROR: ORL P1.#BFFH :TURN OFF MOTOR			
B45C 845C	955	DEAD: JMP TOU DEAD :LOOP BECAUSE SOMETHING IS WRONG			
	956	:			
	957	:THESE ARE ALL SUBROUTINES THAT ARE CALLED:			
	958	:			
B45E 19	959	INCTST: INC T30:OUTBUF :UPDATE THE POINTER			
B45F 237B	960	:GET THE VALUE FOR THE LAST CHARACTER			
B461 D9	961	:ADD THE TEST			
B462 83	962	:EXIT			
B463 89	963	GTPTNT: IN A 00:P1 :READ PORT ONE MOTOR			
B464 37	964	:FLIP BITS			
B465 D263	965	JB6 :GTPTNT :LOOP UNTIL SENSOR IS UNCOVERED			
B467 166B	966	TSTJTF: JTF PIT :SEE IF TIMER FLAG IS SET			
B469 8467	967	:TEST FLAG			
B46B 65	968	PIT: STOP TCNT :STOP THE TIMER			
B46C FF	969	:GET THE CHARACTER			
B46D 34C1	970	:PRINT THE CHARACTER			
B46F 341C	971	:GET ANOTHER CHARACTER			
B471 83	972	:EXIT			
B472 F9	973	DECTST: MOV T30:A:OUTBUF :GET OUTBUF			
B473 87	974	DEC INT A :REDUCE BY ONE			
B474 A9	975	MOV T:OUTBUF:A :PUT BACK IN OUTBUF			
B475 D31F	976	XRL T:A:#FIRST-1 :SEE IF IT IS ALL THE WAY DOWN			
B477 83	977	RETND :EXIT			
	978	:			
	979	:THIS ROUTINE DOES A LINE FEED			
	980	:			
B478 FE	981	LINEF0: MOV A:A:LINCNT :GET THE LINE COUNT			
B479 F29B	982	:IF BIT 7 IS SET, DO A FORMFEED			
B47B 99FD	983	LFD0: ANL T:P1:#BFDH :TURN ON THE SOLENOID			
B47D BC4D	984	:LOAD ONE DELAY			
B47F BF93	985	LFLP1: MOV JUNK1:#3H :LOAD ANOTHER DELAY			
B481 EF81	986	LFLP2: DJNZ JUNK1:LFLP2 :LOOP SOME MORE			
B483 EC7F	987	DJNZ TEMP1:LFLP1 :LOOP SOME MORE			
B485 89B2	988	ORL ON A:P1:#B2H :TURN OFF LF SOLENOID			
B487 1E	989	INC T:LINCNT :UPDATE THE LINE COUNTER			
B488 FE	990	MOV A:A:LINCNT :GET THE LINE COUNT			
B489 D32B	991	XRL T:A:#2BH :IS PAGE DONE			
B48B 96BF	992	JNZ NOTDON :SKIP OVER			
B48D BEBB	993	MOV A:LINCNT:#BBH :ZERO LINE COUNTER			
	994	:			
	995	:NOW DELAY 9B MILLISECDS			
	996	:			
B48F BC8B	997	NOTDON: MOV TEMP1:#3BH :LOAD DELAY VALUES			
B491 BFFF	998	LOP1: MOV JUNK1:#BFFH :			
B493 EF93	999	LOP2: DJNZ JUNK1:LOP2 :GENERATE DELAY			
B495 EC91	1000	DJNZ TEMP1:LOP1 :			
B497 83	1001	RET :LINE FEED IS DONE			
	1002	:			
	1003	:THIS ROUTINE DOES A FORM FEED			
	1004	:			
B498 89	1005	DOFF: IN A:P1 :GET THE STATUS			
B499 37	1006	CPL A :FLIP ACC			
B49A 53CB	1007	ANL A:#BCBH :LEAVE ONLY TWO MSB'S			
B49C C69B	1008	JZ DOFF :IF A FLAG ISN'T COVERED, LOOP			
B49E 89B1	1009	ORL P1:#B1H :TURN THE MOTOR OFF			
B4A0 947B	1010	CALL LFD0 :GO DO ONE LINE FEED			
B4A2 FE	1011	FFCK: MOV A:LINCNT :GET THE LINE COUNT			
B4A3 537F	1012	ANL A:#7FH :STRIP BIT SEVEN			
B4A5 D3BB	1013	XRL A:#BBH :IS IT DONE			
B4A7 C6AD	1014	JZ FFDONE :LEAVE IF IT IS			
B4A9 947B	1015	CALL LFD0 :STROBE THE SOLENOIDS			
B4AB 84A2	1016	JMP FFCK :CHECK THE FORM FEED OUT			
B4AD 83	1017	FFDONE: RET :EXIT FORM FEED			
	1018	:			
B4AE 23EB	1019	SETTIM: MOV A:#EBH :GET DELAY VALUE			
B4B0 62	1020	MOV T:A :PUT IN TIMER			
B4B1 55	1021	STRT T :START THE TIMER			
B4B2 83	1022	RET :EXIT			
	1023	:			

LOC	OBJ	SEQ	SOURCE STATEMENT
B4B3	42	1B24	PRINTBK: MOV A, T :GET THE TIMER
B4B4	37	1B25	CPL T :TWO'S COMPLEMENT ACC
B4B5	17	1B26	INC A
B4B6	17	1B27	INC A
B4B7	17	1B28	INC A
B4B8	17	1B29	INC A
B4B9	17	1B30	INC A
B4BA	62	1B31	MOV T, A :PUT IT BACK IN THE TIMER
B4BB	89	1B32	INLOOP: IN A, T :READ PORT 1
B4BC	F2CB	1B33	J87 COMPBK :IF SENSOR IN NOT COVERED, LEAVE
B4BE	84BB	1B34	JMP INLOOP :OTHERWISE LOOP
B4C0	55	1B35	CONPBK: START T :START THE TIMER
B4C1	16C5	1B36	CONPB: JTF RDTOPT :SEE IF READY TO PRINT
B4C3	84C1	1B37	JMP COMPB :OTHERWISE LOOP
B4C5	23FF	1B38	RDOPT: MOV A, 0BFFH :LOAD A
B4C7	62	1B39	MOV T, A :PUT IT IN THE TIMER
B4C8	93	1B40	RET :EXIT
		1B41	:THIS ROUTINE ADJUSTS AND SAVES THE STATUS DURING PRINTING
		1B42	
		1B43	
B4C9	FD	1B44	STACK: MOV A, STATUS :GET THE STATUS
B4CA	92D2	1B45	J84 LFSET :SET LINE FEED BIT
B4CC	AA	1B46	B4RET: MOV A, SAVPNT, A :SAVE THE STATUS
B4CD	53C2	1B47	ANL A, 0BC2H :RESET EVERYTHING EXCEPT
		1B48	:DIRECTION AND PRINT
B4CF	AD	1B49	MOV STATUS, A :PUT THE STATUS BACK
B4DB	B413	1B50	JMP LPRINT :EXIT
B4D2	432B	1B51	LFSET: ORL A, 02BH :SET BIT 5
B4D4	84CC	1B52	JMP B4RET :JUMP BACK
		1B53	
		1B54	:THIS ROUTINE READS A CHARACTER AND PUTS IT IN THE ACC
		1B55	
B4D6	99EF	1B56	GTCHAR: ANL A, 0BFFH :SET ENABLE BIT
B4D8	88	1B57	MOVX A, 0INBUF :READ THE CHARACTER
B4D9	891B	1B58	ORL A, 01BH :RESET ENABLE BIT
B4DB	83	1B59	RET :EXIT GTCHAR
		1B60	
		1B61	:THIS ROUTINE TURNS THE MOTOR ON
		1B62	
B4DC	99FE	1B63	MOTOR: ANL A, 0BFEH :TURN MOTOR ON
B4DE	83	1B64	RET :EXIT
		1B65	
		1B66	:THIS ROUTINE TURNS THE MOTOR OFF
		1B67	
B4DF	89B1	1B68	MOTOF: ORL A, 0B1H :TURN MOTOR OFF
B4E1	83	1B69	RET :EXIT
		1B70	
		1B71	END :DONE

USER SYMBOLS

ARND	0187	ARNDJP	0149	B4RET	04CC	BACKWARD	0203	CGIN	0400	BACKWARD	0385	BUTLOP	0113	BYERVE	0168
CASE0	0031	CASE01	0017	CASE1	0052	CASE2	0000	CASE23	0024	CASE3	00C2	CHAR	011F	CLPNEM	0448
CONPB	04C1	CONPBK	04C0	CRFIX	01A6	CRFND	00DC	CRFND	0062	CRFD	045C	DECTST	0472	DOFF	0498
DOLF	0071	DONEF	042A	DONEL	043C	DONER	0410	ERROR	045A	FDC	0042	FDC1	0044	FDCR	002E
FDCR1	00A0	FFCK	0442	FFDONE	0440	FFFIX	0102	FINE	0178	FIRE	01CC	FIREX	0108	FIREY	0106
FIRST	0020	FIXDUN	0174	FIXFIN	018F	FIXUP	0189	FXCHAR	0161	FXPRNT	0191	GETSTA	0144	GOOD	0128
GTCHAR	0406	GTPRNT	0463	INBUF	0000	INCTST	045E	INLOOP	0408	ISCHAR	0180	JUNK1	00D7	KTDUN	01E0
LDBUF	0100	LFCKCK	0144	LFDD	0478	LFPIX	01A8	LFPL1	047F	LFPL2	0401	LFSET	0402	LFTEST	017F
LININT	0006	LINEFD	0478	LWI	02A6	LWI11	0389	LKLO	02A6	LKLO1	0207	LWMODE	011C	LOOPW	007A
LOP1	0491	LOP2	0493	LPRINT	0011	LPRINT1	0013	MAX	006F	MOTOF	040F	MOTON	040C	NOFF	018F
NOLF	011A	NOTDOW	048F	HT1	01D4	OUTBUF	0001	OVR	0004	OVR1	0005	PAGE1	02A0	PAGE2	03A0
PIT	0468	PRNT	0084	PNTBK	0483	PNTIT	01C1	POTOPT	04C5	SAVPNT	0002	SELF	041F	SELF1	0421
SELFA	0411	SELFAA	0430	SELF	048F	SELFBB	042E	SELF	0400	SELFCC	042C	SETTIM	044E	SETUP	0400
SHORT	01CA	STACK	04C9	STATUS	0005	STCNT	0003	STBIT1	0150	STPRNT	0159	SUB1	0139	TAB1	0200
TEMP1	0004	TSJTF	01DC	TSJTF	0463	VARSET	043F	WATCH	0075	WATCHD	00NE				

ASSEMBLY COMPLETE. NO ERRORS

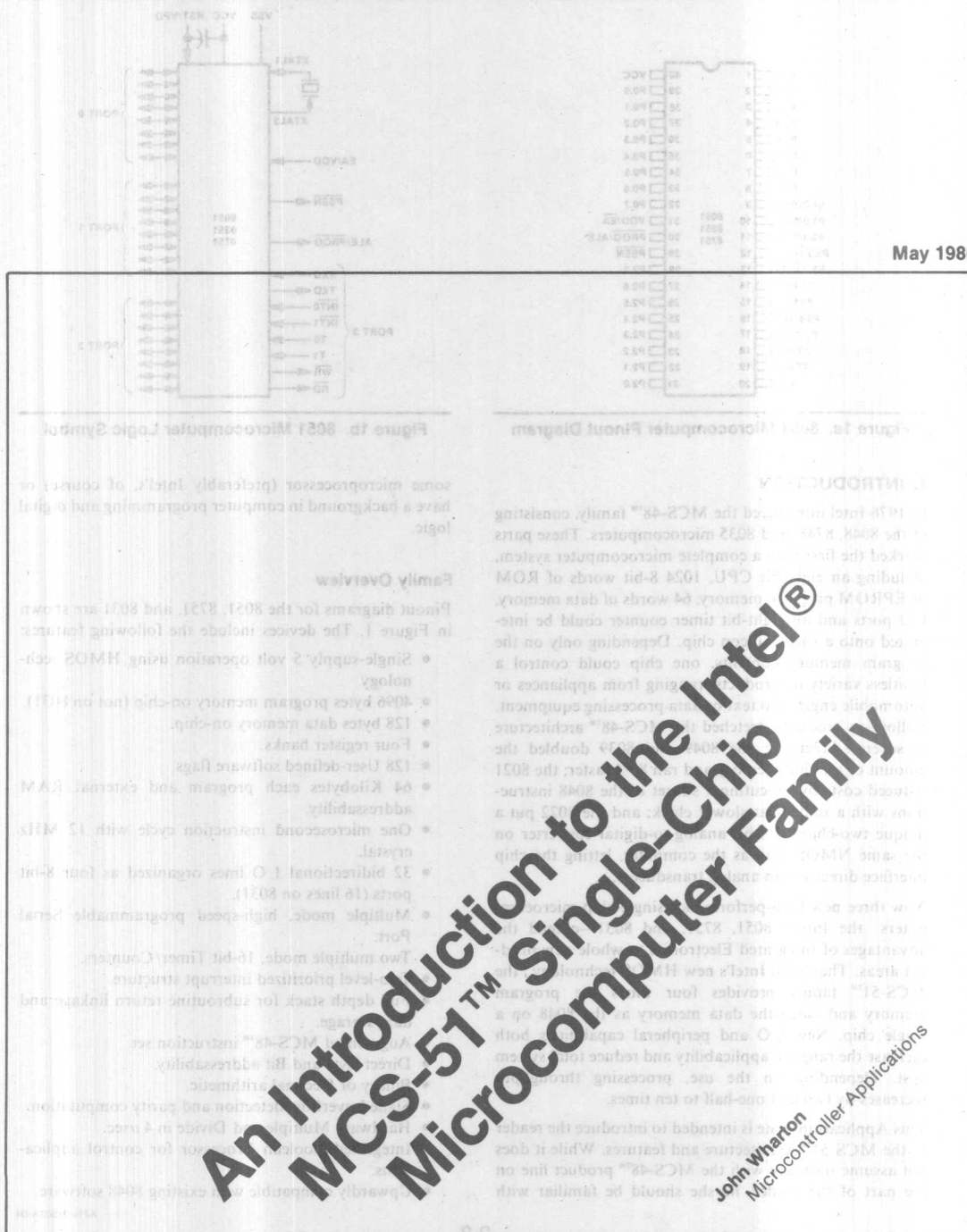
MCS®-51 Application Notes & Article Reprints

2

2

MCS®-51 Application Notes &
Article Reprints

May 1980



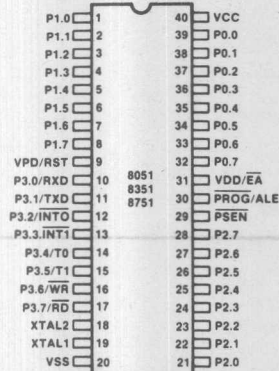


Figure 1a. 8051 Microcomputer Pinout Diagram

1. INTRODUCTION

In 1976 Intel introduced the MCS-48™ family, consisting of the 8048, 8748, and 8035 microcomputers. These parts marked the first time a complete microcomputer system, including an eight-bit CPU, 1024 8-bit words of ROM or EPROM program memory, 64 words of data memory, I/O ports and an eight-bit timer counter could be integrated onto a single silicon chip. Depending only on the program memory contents, one chip could control a limitless variety of products, ranging from appliances or automobile engines to text or data processing equipment. Follow-on products stretched the MCS-48™ architecture in several directions: the 8049 and 8039 doubled the amount of on-chip memory and ran 83% faster; the 8021 reduced costs by executing a subset of the 8048 instructions with a somewhat slower clock; and the 8022 put a unique two-channel 8-bit analog-to-digital converter on the same NMOS chip as the computer, letting the chip interface directly with analog transducers.

Now three new high-performance single-chip microcomputers—the Intel® 8051, 8751, and 8031—extend the advantages of Integrated Electronics to whole new product areas. Thanks to Intel's new HMOS technology, the MCS-51™ family provides four times the program memory and twice the data memory as the 8048 on a single chip. New I/O and peripheral capabilities both increase the range of applicability and reduce total system cost. Depending on the use, processing throughput increases by two and one-half to ten times.

This Application Note is intended to introduce the reader to the MCS-51™ architecture and features. While it does not assume intimacy with the MCS-48™ product line on the part of the reader, he/she should be familiar with

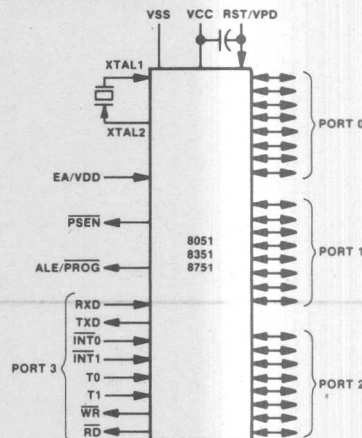


Figure 1b. 8051 Microcomputer Logic Symbol

some microprocessor (preferably Intel's, of course) or have a background in computer programming and digital logic.

Family Overview

Pinout diagrams for the 8051, 8751, and 8031 are shown in Figure 1. The devices include the following features:

- Single-supply 5 volt operation using HMOS technology.
- 4096 bytes program memory on-chip (not on 8031).
- 128 bytes data memory on-chip.
- Four register banks.
- 128 User-defined software flags.
- 64 Kilobytes each program and external RAM addressability.
- One microsecond instruction cycle with 12 MHz crystal.
- 32 bidirectional I/O lines organized as four 8-bit ports (16 lines on 8031).
- Multiple mode, high-speed programmable Serial Port.
- Two multiple mode, 16-bit Timer/Counters.
- Two-level prioritized interrupt structure.
- Full depth stack for subroutine return linkage and data storage.
- Augmented MCS-48™ instruction set.
- Direct Byte and Bit addressability.
- Binary or Decimal arithmetic.
- Signed-overflow detection and parity computation.
- Hardware Multiple and Divide in 4 μ sec.
- Integrated Boolean Processor for control applications.
- Upwardly compatible with existing 8048 software.

AFN-01502A-04

All three devices come in a standard 40-pin Dual In-Line Package, with the same pin-out, the same timing, and the same electrical characteristics. The primary difference between the three is the on-chip program memory—different types are offered to satisfy differing user requirements.

The 8751 provides 4K bytes of ultraviolet-Erasable, Programmable Read Only Memory (EPROM) for program development, prototyping, and limited production runs. (By convention, 1K means $2^{10} = 1024$. 1k—with a lower case “k”—equals $10^3 = 1000$.) This part may be individually programmed for a specific application using Intel's Universal PROM Programmer (UPP). If software bugs are detected or design specifications change the same part may be “erased” in a matter of minutes by exposure to ultraviolet light and reprogrammed with the modified code. This cycle may be repeated indefinitely during the design and development phase.

The final version of the software must be programmed into a large number of production parts. The 8051 has 4K bytes of ROM which are mask-programmed with the customer's order when the chip is built. This part is considerably less expensive, but cannot be erased or altered after fabrication.

The 8031 does not have any program memory on-chip, but may be used with up to 64K bytes of external standard or multiplexed ROMs, PROMs, or EPROMs. The 8031 fits well in applications requiring significantly larger or smaller amounts of memory than the 4K bytes provided by its two siblings.

(The 8051 and 8751 automatically access external program memory for all addresses greater than the 4096 bytes on-chip. The External Access input is an override for all internal program memory—the 8051 and 8751 will each emulate an 8031 when pin 31 is low.)

Throughout this Note, “8051” is used as a generic term. Unless specifically stated otherwise, the point applies equally to all three components. Table 1 summarizes the quantitative differences between the members of the MCS-48™ and MCS-51™ families.

The remainder of this Note discusses the various MCS-51™ features and how they can be used. Software and/or hard-

ware application examples illustrate many of the concepts. Several isolated tasks (rather than one complete system design example) are presented in the hope that some of them will apply to the reader's experiences or needs.

A document this short cannot detail all of a computer system's capabilities. By no means will all the 8051 instructions be demonstrated; the intent is to stress new or unique MCS-51™ operations and instructions generally used in conjunction with each other. For additional hardware information refer to the Intel MCS-51™ Family User's Manual, publication number 121517. The assembly language and use of ASM51, the MCS-51™ assembler, are further described in the MCS-51™ Macro Assembler User's Guide, publication number 9800937.

The next section reviews some of the basic concepts of microcomputer design and use. Readers familiar with the 8048 may wish to skim through this section or skip directly to the next, “ARCHITECTURE AND ORGANIZATION.”

Microcomputer Background Concepts

Most digital computers use the binary (base 2) number system internally. All variables, constants, alphanumeric characters, program statements, etc., are represented by groups of binary digits (“bits”), each of which has the value 0 or 1. Computers are classified by how many bits they can move or process at a time.

The MCS-51™ microcomputers contain an eight-bit central processing unit (CPU). Most operations process variables eight bits wide. All internal RAM and ROM, and virtually all other registers are also eight bits wide. An eight-bit (“byte”) variable (shown in Figure 2) may assume one of $2^8 = 256$ distinct values, which usually represent integers between 0 and 255. Other types of numbers, instructions, and so forth are represented by one or more bytes using certain conventions.

For example, to represent positive and negative values, the most significant bit (D7) indicates the sign of the other seven bits—0 if positive, 1 if negative—allowing integer variables between -128 and +127. For integers with extremely large magnitudes, several bytes are manipulated together as “multiple precision” signed or unsigned integers—16, 24, or more bits wide.

Table 1. Features of Intel's Single-Chip Microcomputers

EPROM Program Memory	ROM Program Memory	External Program Memory	Program Memory (Int/Max)	Data Memory (Bytes)	Instr. Cycle Time	Input/Output Pins	Interrupt Sources	Reg. Banks
	8021	—	1K 1K	64	8.4 μ Sec	21	0	1
	8022	—	2K 2K	64	8.4 μ Sec	28	2	1
8748	8048	8035	1K 4K	64	2.5 μ Sec	27	2	2
	8049	8039	2K 4K	128	1.36 μ Sec	27	2	2
8751	8051	8031	4K 64K	128	1.0 μ Sec	32	5	4

The letters "MCS" have traditionally indicated a system or family of compatible Intel® microcomputer components, including CPUs, memories, clock generators, I/O expanders, and so forth. The numerical suffix indicates the microprocessor or microcomputer which serves as the cornerstone of the family. Microcomputers in the MCS-48™ family currently include the 8048-series (8035, 8048, & 8748), the 8049-series (8039 & 8049), and the 8021 and 8022; the family also includes the 8243, an I/O expander compatible with each of the microcomputers. Each computer's CPU is derived from the 8048, with essentially the same architecture, addressing modes, and instruction set, and a single assembler (ASM48) serves each.

The first members of the MCS-51™ family are the 8051, 8751, and 8031. The architecture of the 8051-series, while derived from the 8048, is not strictly compatible; there are more addressing modes, more instructions, larger address spaces, and a few other hardware differences. In this Application Note the letters "MCS-51" are used when referring to architectural features of the 8051-series—features which would be included on possible future microcomputers based on the 8051 CPU. Such products could have different amounts of memory (as in the 8048/8049) or different peripheral functions (as in the 8021 and 8022) while leaving the CPU and instruction set intact. ASM51 is the assembler used by all microcomputers in the 8051 family.

Two digit decimal numbers may be "packed" in an eight-bit value, using four bits for the binary code of each digit. This is called Binary-Coded Decimal (BCD) representation, and is often used internally in programs which interact heavily with human beings.

Alphanumeric characters (letters, numbers, punctuation marks, etc.) are often represented using the American Standard Code for Information Interchange (ASCII) convention. Each character is associated with a unique seven-bit binary number. Thus one byte may represent

0	1	0	1	0	0	0	1
D7	D6	D5	D4	D3	D2	D1	D0

Figure 2. Representation of Bits Within an Eight-Bit "Byte" (Value shown = 01010001 Binary = 81 decimal).

a single character, and a word or sequence of letters may be represented by a series (or "string") of bytes. Since the ASCII code only uses 128 characters, the most significant bit of the byte is not needed to distinguish between characters. Often D7 is set to 0 for all characters. In some coding schemes, D7 is used to indicate the "parity" of the other seven bits—set or cleared as necessary to ensure that the total number of "1" bits in the eight-bit code is even ("even parity") or odd ("odd parity"). The 8051 includes hardware to compute parity when it is needed.

A computer program consists of an ordered sequence of specific, simple steps to be executed by the CPU one-at-a-time. The method or sequence of steps used collectively to solve the user's application is called an "algorithm."

The program is stored inside the computer as a sequence of binary numbers, where each number corresponds to one of the basic operations ("opcodes") which the CPU is capable of executing. In the 8051, each program memory location is one byte. A complete instruction consists of a sequence of one or more bytes, where the first defines the operation to be executed and additional bytes (if needed) hold additional information, such as data values or variable addresses. No instruction is longer than three bytes.

The way in which binary opcodes and modifier bytes are assigned to the CPU's operations is called the computer's "machine language." Writing a program directly in machine language is time-consuming and tedious. Human beings think in words and concepts rather than encoded numbers, so each CPU operation and resource is given a name and standard abbreviation ("mnemonic"). Programs are more easily discussed using these standard mnemonics, or "assembly language," and may be typed into an Intel® Intellec® 800 or Series II® microcomputer development system in this form. The development system can mechanically translate the program from assembly language "source" form to machine language "object" code using a program called an "assembler." The MCS-51™ assembler is called ASM51.

There are several important differences between a computer's machine language and the assembly language used as a tool to represent it. The machine language or instruction set is the set of operations which the CPU can perform while a program is executing ("at run-time"), and is strictly determined by the microcomputer hardware design.

The assembly language is a standard (though more-or-less arbitrary) set of symbols including the instruction set mnemonics, but with additional features which further simplify the program design process. For example, ASM51 has controls for creating and formatting a program listing, and a number of directives for allocating variable storage and inserting arbitrary bytes of data into the object code for creating tables of constants.

In addition, ASM51 can perform sophisticated mathematical operations, computing addresses or evaluating arithmetic expressions to relieve the programmer from this drudgery. However, these calculations can only use information known at "assembly time."

For example, the 8051 performs arithmetic calculations at run-time, eight bits at a time. ASM51 can do similar operations 16 bits at a time. The 8051 can only do one simple step per instruction, while ASM51 can perform complex calculations in each line of source code. However, the operations performed by the assembler may only use parameter values fixed at assembly-time, not variables whose values are unknown until program execution begins.

For example, when the assembly language source line,

```
ADD A,#(LOOP_COUNT + 1) * 3
```

is assembled, ASM51 will find the value of the previously-defined constant "LOOP_COUNT" in an internal symbol table, increment the value, multiply the sum by three, and (assuming it is between -256 and 255 inclusive) truncate the product to eight bits. When this instruction is executed, the 8051 ALU will just add that resulting constant to the accumulator.

Some similar differences exist to distinguish number system ("radix") specifications. The 8051 does all computations in binary (though there are provisions for then converting the result to decimal form). In the course of writing a program, though, it may be more convenient to specify constants using some other radix, such as base 10. On other occasions, it is desirable to specify the ASCII code for some character or string of characters without referring to tables. ASM51 allows several representations for constants, which are converted to binary as each instruction is assembled.

For example, binary numbers are represented in the

assembly language by a series of ones and zeros (naturally), followed by the letter "B" (for Binary); octal numbers as a series of octal digits (0-7) followed by the letter "O" (for Octal) or "Q" (which doesn't stand for anything, but *looks* sort of like an "O" and is less likely to be confused with a zero).

Hexadecimal numbers are represented by a series of hexadecimal digits (0-9,A-F), followed by (you guessed it) the letter "H." A "hex" number must begin with a decimal digit; otherwise it would look like a user-defined symbol (to be discussed later). A "dummy" leading zero may be inserted before the first digit to meet this constraint. The character string "BACH" could be a legal label for a Baroque music synthesis routine; the string "0BACH" is the hexadecimal constant BACH₁₆. This is a case where adding 0 makes a big difference.

Decimal numbers are represented by a sequence of decimal digits, optionally followed by a "D." If a number has no suffix, it is assumed to be decimal—so it had better not contain any non-decimal digits. "0BAC" is not a legal representation for anything.

When an ASCII code is needed in a program, enclose the desired character between two apostrophes (as in '#') and the assembler will convert it to the appropriate code (in this case 23H). A string of characters between apostrophes is translated into a series of constants; 'BACH' becomes 42H, 41H, 43H, 48H.

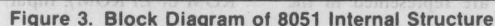
These same conventions are used throughout the associated Intel documentation. Table 2 illustrates some of the different number formats.

2. ARCHITECTURE AND ORGANIZATION

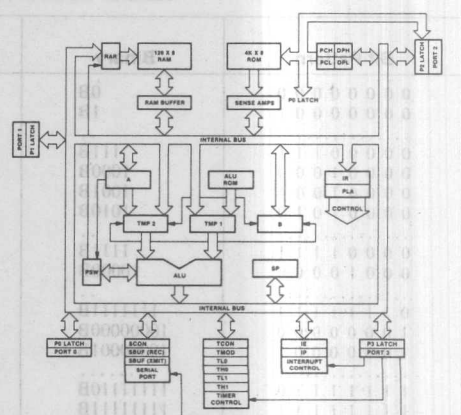
Figure 3 blocks out the MCS-51™ internal organization. Each microcomputer combines a Central Processing Unit, two kinds of memory (data RAM plus program ROM or EPROM), Input/Output ports, and the mode,

Table 2. Notations Used to Represent Numbers

Bit Pattern	Binary	Octal	Hexa-Decimal	Decimal	Signed Decimal
0 0 0 0 0 0 0 0	0B	0Q	00H	0	0
0 0 0 0 0 0 0 1	1B	1Q	01H	1	+1
...
0 0 0 0 0 1 1 1	111B	7Q	07H	7	+7
0 0 0 0 1 0 0 0	1000B	10Q	08H	8	+8
0 0 0 0 1 0 0 1	1001B	11Q	09H	9	+9
0 0 0 0 1 0 1 0	1010B	12Q	0AH	10	+10
...
0 0 0 0 1 1 1 1	1111B	17Q	0FH	15	+15
0 0 0 1 0 0 0 0	10000B	20Q	10H	16	+16
...
0 1 1 1 1 1 1 1	111111B	177Q	7FH	127	+127
1 0 0 0 0 0 0 0	1000000B	200Q	80H	128	+128
1 0 0 0 0 0 0 1	1000001B	201Q	81H	129	+129
...
1 1 1 1 1 1 1 0	11111110B	376Q	0FEH	254	-2
1 1 1 1 1 1 1 1	11111111B	377Q	0FFH	255	-1



The CPU is the "brains" of the microcomputer, reading the user's program and executing the instructions stored therein. Its primary elements are an eight-bit Arithmetic/Logic Unit with associated registers A, B, PSW, and SP, and the sixteen-bit Program Counter and "Data Pointer" registers.



Arithmetic Logic Unit

The ALU can perform (as the name implies) arithmetic and logic functions on eight-bit variables. The former include basic addition, subtraction, multiplication, and division; the latter include the logical operations AND, OR, and Exclusive-OR, as well as rotate, clear, complement, and so forth. The ALU also makes conditional branching decisions, and provides data paths and temporary registers used for data transfers within the system. Other instructions are built up from these primitive functions: the addition capability can increment registers or automatically compute program destination addresses; subtraction is also used in decrementing or comparing the magnitude of two variables.

These primitive operations are automatically cascaded and combined with dedicated logic to build complex instructions such as incrementing a sixteen-bit register pair. To execute one form of the compare instruction, for example, the 8051 increments the program counter three times, reads three bytes of program memory, computes a register address with logical operations, reads internal data memory twice, makes an arithmetic comparison of two variables, computes a sixteen-bit destination address, and decides whether or not to make a branch—all in two microseconds!

An important and unique feature of the MCS-51 architecture is that the ALU can also manipulate one-bit as well as eight-bit data types. Individual bits may be set, cleared, or complemented, moved, tested, and used in logic computations. While support for a more primitive data type may initially seem a step backwards in an era of increasing word length, it makes the 8051 especially well suited for controller-type applications. Such algorithms inherently involve Boolean (true/false) input and output variables, which were heretofore difficult to implement with standard microprocessors. These features are collectively referred to as the MCS-51™ "Boolean Processor," and are described in the so-named chapter to come.

Thanks to this powerful ALU, the 8051 instruction set fares well at both real-time control and data intensive algorithms. A total of 51 separate operations move and manipulate three data types: Boolean (1-bit), byte (8-bit), and address (16-bit). All told, there are eleven addressing modes—seven for data, four for program sequence control (though only eight are used by more than just a few specialized instructions). Most operations allow several addressing modes, bringing the total number of instructions (operation/addressing mode combinations) to 111, encompassing 255 of the 256 possible eight-bit instruction opcodes.

Instruction Set Overview

Table 4 lists these 111 instructions, classified into five groups:

- Arithmetic Operations
- Logical Operations for Byte Variables
- Data Transfer Instructions
- Boolean Variable Manipulation
- Program Branching and Machine Control

MCS-48™ programmers perusing Table 4 will notice the absence of special categories for Input/Output, Timer/Counter, or Control instructions. These functions are all still provided (and indeed many new functions are added), but as special cases of more generalized operations in other categories. To explicitly list all the useful instructions involving I/O and peripheral registers would require a table approximately four times as long.

Observant readers will also notice that all of the 8048's page-oriented instructions (conditional jumps, JMPP, MOVP, MOVP3) have been replaced with corresponding but non-paged instructions. The 8051 instruction set is entirely *non-page-oriented*. The MCS-48™ "MOVP" instruction replacement and all conditional jump instructions operate relative to the program counter, with the actual jump address computed by the CPU during instruction execution. The "MOVP3" and "JMPP" replacements are now made relative to another sixteen-bit register, which allows the effective destination to be anywhere in the program memory space, regardless of where the instruction itself is located. There are even three-byte jump and call instructions allowing the destination to be *anywhere* in the 64K program address space.

The instruction set is designed to make programs efficient both in terms of code size and execution speed. No instruction requires more than three bytes of program memory, with the majority requiring only one or two bytes. Virtually all instructions execute in either one or two instruction cycles—one or two microseconds with a 12-MHz crystal—with the sole exceptions (multiply and divide) completing in four cycles.

Many instructions such as arithmetic and logical functions or program control, provide both a short and a long form for the same operation, allowing the programmer to optimize the code produced for a specific application. The 8051 usually fetches two instruction bytes per instruction cycle, so using a shorter form can lead to faster execution as well.

For example, any byte of RAM may be loaded with a constant with a three-byte, two-cycle instruction, but the commonly used "working registers" in RAM may be initialized in one cycle with a two-byte form. Any bit anywhere on the chip may be set, cleared, or complemented by a single three-byte logical instruction using two cycles. But critical control bits, I/O pins, and software flags may be controlled by two-byte, single cycle instructions. While three-byte jumps and calls can "go anywhere" in program memory, nearby sections of code may be reached by shorter relative or absolute versions.

(MSB)				(LSB)			
CY	AC	F0	RS1	RS0	OV	—	P

Symbol Position Name and Significance

CY	PSW.7	Carry flag. Set/cleared by hardware or software during certain arithmetic and logical instructions.
AC	PSW.6	Auxiliary Carry flag. Set/cleared by hardware during addition or subtraction instructions to indicate carry or borrow out of bit 3.
F0	PSW.5	Flag 0 Set/cleared/tested by software as a user-defined status flag.
RS1	PSW.4	Register bank Select control bits 1 & 0. Set/cleared by software to determine working register bank (see Note).
RS	PSW.3	

Symbol Position Name and Significance

OV	PSW.2	Overflow flag. Set/cleared by hardware during arithmetic instructions to indicate overflow conditions.
----	-------	---

— PSW.1 (reserved)

P	PSW.0	Parity flag. Set/cleared by hardware each instruction cycle to indicate an odd/even number of "one" bits in the accumulator, i.e., even parity.
---	-------	--

Note— the contents of (RS1, RS0) enable the working register banks as follows:

(0,0) — Bank 0	(00H-07H)
(0,1) — Bank 1	(08H-0FH)
(1,0) — Bank 2	(10H-17H)
(1,1) — Bank 3	(18H-1FH)

Figure 4. PSW—Program Status Word Organization

A significant side benefit of an instruction set more powerful than those of previous single-chip microcomputers is that it is easier to generate applications-oriented software. Generalized addressing modes for byte and bit instructions reduce the number of source code lines written and debugged for a given application. This leads in turn to proportionately lower software costs, greater reliability, and faster design cycles.

Accumulator and PSW

The 8051, like its 8048 predecessor, is primarily an accumulator-based architecture: an eight-bit register called the accumulator ("A") holds a source operand and receives the result of the arithmetic instructions (addition, subtraction, multiplication, and division). The accumulator can be the source or destination for logical operations and a number of special data movement instructions, including table look-ups and external RAM expansion. Several functions apply exclusively to the accumulator: rotates, parity computation, testing for zero, and so on.

Many instructions implicitly or explicitly affect (or are affected by) several status flags, which are grouped together to form the Program Status Word shown in Figure 4.

(The period within entries under the Position column is called the "dot operator," and indicates a particular bit position within an eight-bit byte. "PSW.5" specifies bit 5 of the PSW. Both the documentation and ASM51 use this notation.)

The most "active" status bit is called the carry flag (abbreviated "C"). This bit makes possible multiple precision arithmetic operations including addition, subtraction,

and rotates. The carry also serves as a "Boolean accumulator" for one-bit logical operations and bit manipulation instructions. The overflow flag (OV) detects when arithmetic overflow occurs on signed integer operands, making two's complement arithmetic possible. The parity flag (P) is updated after every instruction cycle with the even-odd parity of the accumulator contents.

The CPU does not control the two register-bank select bits, RS1 and RS0. Rather, they are manipulated by software to enable one of the four register banks. The usage of the PSW flags is demonstrated in the Instruction Set chapter of this Note.

Even though the architecture is accumulator-based, provisions have been made to bypass the accumulator in common instruction situations. Data may be moved from any location on-chip to any register, address, or indirect address (and vice versa), any register may be loaded with a constant, etc., all without affecting the accumulator. Logical operations may be performed against registers or variables to alter fields of bits—without using or affecting the accumulator. Variables may be incremented, decremented, or tested without using the accumulator. Flags and control bits may be manipulated and tested without affecting anything else.

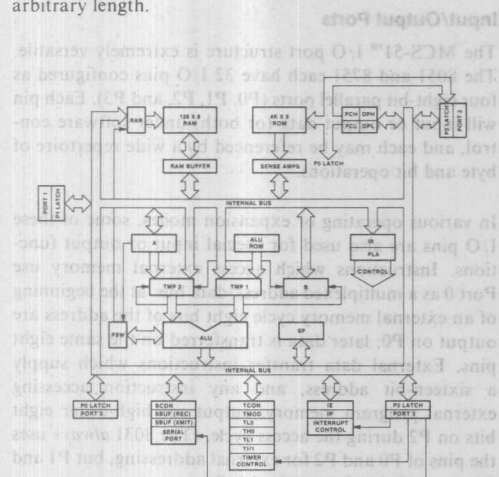
Other CPU Registers

A special eight-bit register ("B") serves in the execution of the multiply and divide instructions. This register is used in conjunction with the accumulator as the second input operand and to return eight-bits of the result.

The MCS-51 family processors include a hardware stack within internal RAM, useful for subroutine linkage.

passing parameters between routines, temporary variable storage, or saving status during interrupt service routines. The Stack Pointer (SP) is an eight-bit pointer register which indicates the address of the last byte pushed onto the stack. The stack pointer is automatically incremented or decremented on all push or pop instructions and all subroutine calls and returns. In theory, the stack in the 8051 may be up to a full 128 bytes deep. (In practice, even simple programs would use a handful of RAM locations for pointers, variables, and so forth—reducing the stack depth by that number.) The stack pointer defaults to 7 on reset, so that the stack will start growing up from location 8, just like in the 8048. By altering the pointer contents the stack may be relocated anywhere within internal RAM.

Finally, a 16-bit register called the data pointer (DPTR) serves as a base register in indirect jumps, table look-up instructions, and external data transfers. The high- and low-order halves of the data pointer may be manipulated as separate registers (DPH and DPL, respectively) or together using special instructions to load or increment all sixteen bits. Unlike the 8048, look-up tables can therefore start anywhere in program memory and be of arbitrary length.



Memory Spaces

Program memory is separate and distinct from data memory. Each memory type has a different addressing mechanism, different control signals, and a different function.

The program memory array (ROM or EPROM), like an elephant, is extremely large and never forgets information, even when power is removed. Program memory is used for information needed each time power is applied: initialization values, calibration constants, keyboard layout tables, etc., as well as the program itself. The program memory has a sixteen-bit address bus; its elements

are addressed using the Program Counter or instructions which generate a sixteen-bit address.

To stretch our analogy just a bit, data memory is like a mouse: it is smaller and therefore quicker than program memory, and it goes into a random state when electrical power is applied. On-chip data RAM is used for variables which are determined or may change while the program is running.

A computer spends most of its time manipulating variables, not constants, and a relatively small number of variables at that. Since eight-bits is more than sufficient to uniquely address 128 RAM locations, the on-chip RAM address register is only one byte wide. In contrast to the program memory, data memory accesses need a single eight-bit value—a constant or another variable—to specify a unique location. Since this is the basic width of the ALU and the different memory types, those resources can be used by the addressing mechanisms, contributing greatly to the computer's operating efficiency.

The partitioning of program and data memory is extended to off-chip memory expansion. Each may be added independently, and each uses the same address and data busses, but with different control signals. External program memory is gated onto the external data bus by the PSEN (Program Store Enable) control output, pin 29. External data memory is read onto the bus by the RD output, pin 17, and written with data supplied from the microcomputer by the WR output, pin 16. (There is no control pin to write external program ROM, which is by definition Read Only.) While both types may be expanded to up to 64K bytes, the external data memory may optionally be expanded in 256 byte "pages" to preserve the use of P2 as an I/O port. This is useful with a relatively small expansion RAM (such as the Intel® 8155) or for addressing external peripherals.

Single-chip controller programs are finalized during the project design cycle, and are not modified after production. Intel's single-chip microcomputers are not "von Neumann" architectures common among main-frame and mini-computer systems: the MCS-51™ processor data memory—on-chip and external—may not be used for program code. Just as there is no write-control signal for program memory, there is no way for the CPU to execute instructions out of RAM. In return, this concession allows an architecture optimized for efficient controller applications: a large, fixed program located in ROM, a hundred or so variables in RAM, and different methods for efficiently addressing each.

(Von Neumann machines are helpful for software development and debug. An 8051 system could be modified to have a single off-chip memory space by gating together the two memory-read controls (PSEN and RD) with a two-input AND gate (Figure 5). The CPU could then write data into the common memory array using WR and

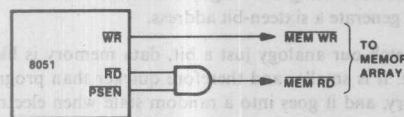


Figure 5. Combining External Program and Data Memory Arrays

external data transfer instructions, and read instructions or data with the AND gate output and data transfer or program memory look-up instructions.)

In addition to the memory arrays, there is (yet) another (albeit sparsely populated) physical address space. Connected to the internal data bus are a score of special-purpose eight-bit registers scattered throughout the chip. Some of these—B, SP, PSW, DPH, and DPL—have been discussed above. Others—I/O ports and peripheral function registers—will be introduced in the following sections. Collectively, these registers are designated as the “special-function register” address space. Even the accumulator is assigned a spot in the special-function register address space for additional flexibility and uniformity.

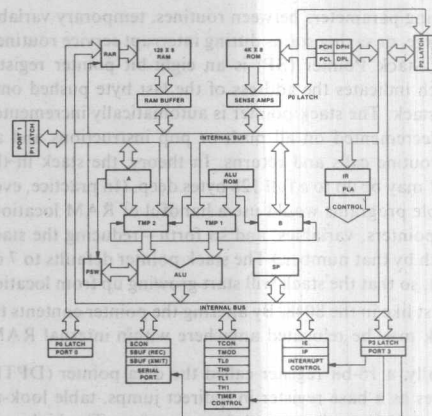
Thus, the MCS-51™ architecture supports several distinct "physical" address spaces, functionally separated at the hardware level by different addressing mechanisms, read and write control signals, or both:

- On-chip program memory;
- On-chip data memory;
- Off-chip program memory;
- Off-chip data memory;
- On-chip special-function registers.

What the *programmer sees*, though, are "logical" address spaces. For example, as far as the programmer is concerned, there is only one type of program memory, 64K bytes in length. The fact that it is formed by combining on- and off-chip arrays (split 4K/60K on the 8051 and 8751) is "invisible" to the programmer; the CPU automatically fetches each byte from the appropriate array, based on its address.

(Presumably, future microcomputers based on the MCS-51™ architecture may have a different physical split, with more or less of the 64K total implemented on-chip. Using the MCS-48™ family as a precedent, the 8048's 4K potential program address space was split 1K/3K between on- and off-chip arrays; the 8049's was split 2K/2K.)

Why go into such tedious details about address spaces? The logical addressing modes are described in the Instruction Set chapter in terms of physical address spaces. Understanding their differences now will pay off in understanding and using the chips later.



Input/Output Ports

The MCS-51[®] I/O port structure is extremely versatile. The 8051 and 8751 each have 32 I/O pins configured as four eight-bit parallel ports (P0, P1, P2, and P3). Each pin will input or output data (or both) under software control, and each may be referenced by a wide repertoire of byte and bit operations.

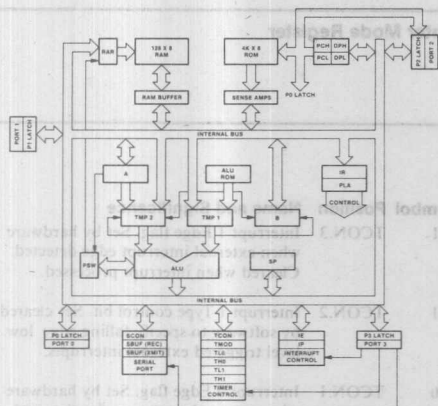
In various operating or expansion modes, some of these I/O pins are also used for special input or output functions. Instructions which access external memory use Port 0 as a multiplexed address/data bus: at the beginning of an external memory cycle eight bits of the address are output on P0; later data is transferred on the same eight pins. External data transfer instructions which supply a sixteen-bit address, and any instruction accessing external program memory, output the high-order eight bits on P2 during the access cycle. (The 8031 *always* uses the pins of P0 and P2 for external addressing, but P1 and P3 are available for standard I/O.)

The eight pins of Port 3 (P3) each have a special function. Two external interrupts, two counter inputs, two serial data lines, and two timing control strobes use pins of P3 as described in Figure 6. Port 3 pins corresponding to functions not used are available for conventional I/O.

Even within a single port, I/O functions may be combined in many ways: input and output may be performed using different pins at the same time, or the same pins at different times; in parallel in some cases, and in serial in others; as test pins, or (in the case of Port 3) as additional special functions.

(MSB)				(LSB)			
RD	WR	T1	T0	INT1	INT0	TXD	RXD
Symbol	Position	Name and Significance					
RD	P3.7	Read data control output. Active low pulse generated by hardware when external data memory is read.					
WR	P3.6	Write data control output. Active low pulse generated by hardware when external data memory is written.					
T1	P3.5	Timer/counter 1 external input or test pin.					
T0	P3.4	Timer/counter 0 external input or test pin.					
Symbol	Position	Name and Significance					
INT1	P3.3	Interrupt 1 input pin. Low-level or falling-edge triggered.					
INT0	P3.2	Interrupt 0 input pin. Low-level or falling-edge triggered.					
TXD	P3.1	Transmit Data pin for serial port in UART mode. Clock output in shift register mode.					
RXD	P3.0	Receive Data pin for serial port in UART mode. Data I/O pin in shift register mode.					

Figure 6. P3—Alternate Special Functions of Port 3



Special Peripheral Functions

There are a few special needs common among control-oriented computer systems:

- keeping track of elapsed real-time;
- maintaining a count of signal transitions;
- measuring the precise width of input pulses;
- communicating with other systems or people;
- closely monitoring asynchronous external events.

Until now, microprocessor systems needed peripheral chips such as timer/counters, USARTs, or interrupt controllers to meet these needs. The 8051 integrates all of these capabilities on-chip!

Timer/Counters

There are two sixteen-bit multiple-mode Timer/Counters on the 8051, each consisting of a "High" byte (corresponding to the 8048 "T" register) and a low byte (similar to the 8048 prescaler, with the additional flexibility of being

software-accessible). These registers are called, naturally enough, TH0, TL0, TH1, and TL1. Each pair may be independently software programmed to any of a dozen modes with a mode register designated TMOD (Figure 7), and controlled with register TCON (Figure 8).

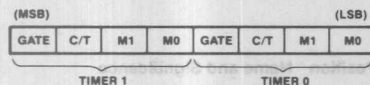
The timer modes can be used to measure time intervals, determine pulse widths, or initiate events, with one-microsecond resolution, up to a maximum interval of 65,536 instruction cycles (over 65 milliseconds). Longer delays may easily be accumulated through software. Configured as a counter, the same hardware will accumulate external events at frequencies from D.C. to 500 KHz, with up to sixteen bits of precision.

Serial Port Interface

Each microcomputer contains a high-speed, full-duplex, serial port which is software programmable to function in four basic modes; shift-register I/O expander, 8-bit UART, 9-bit UART, or interprocessor communications link. The UART modes will interface with standard I/O devices (e.g. CRTs, teletypewriters, or modems) at data rates from 122 baud to 31 kilobaud. Replacing the standard 12 MHz crystal with a 10.7 MHz crystal allows 110 baud. Even or odd parity (if desired) can be included with simple bit-handling software routines. Inter-processor communications in distributed systems takes place at 187 kilobaud with hardware for automatic address/data message recognition. Simple TTL or CMOS shift registers provide low-cost I/O expansion at a super-fast 1 Megabaud. The serial port operating modes are controlled by the contents of register SCON (Figure 9).

Interrupt Capability and Control

(Interrupt capability is generally considered a CPU function. It is being introduced here since, from an applications point of view, interrupts relate more closely to peripheral and system interfacing.)

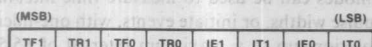


GATE Gating control. When set, Timer/counter "x" is enabled only while "INTx" pin is high and "TRx" control bit is set. When cleared, timer/counter is enabled whenever "TRx" control bit is set.

C/T Timer or Counter Selector. Cleared for Timer operation (input from internal system clock). Set for Counter operation (input from "Tx" input pin).

M1	M0	Operating Mode
0	0	MCS-48 Timer. "TLx" serves as five-bit prescaler.
0	1	16-bit timer counter. "THx" and "TLx" are cascaded; there is no prescaler.
1	0	8-bit auto-reload timer counter. "THx" holds a value which is to be reloaded into "TLx" each time it overflows.
1	1	(Timer 0) TL0 is an eight-bit timer counter controlled by the standard Timer 0 control bits. TH0 is an eight-bit timer only controlled by Timer 1 control bits.
1	1	(Timer 1) Timer counter 1 stopped.

Figure 7. TMOD—Timer/Counter Mode Register



Symbol	Position	Name and Significance
TF1	TCON.7	Timer 1 overflow Flag. Set by hardware on timer/counter overflow. Cleared when interrupt processed.
TR1	TCON.6	Timer 1 Run control bit. Set/cleared by software to turn timer/counter on/off.
TF0	TCON.5	Timer 0 overflow Flag. Set by hardware on timer/counter overflow. Cleared when interrupt processed.
TR0	TCON.4	Timer 0 Run control bit. Set/cleared by software to turn timer/counter on/off.

Symbol	Position	Name and Significance
IE1	TCON.3	Interrupt 1 Edge flag. Set by hardware when external interrupt edge detected. Cleared when interrupt processed.
IT1	TCON.2	Interrupt 1 Type control bit. Set cleared by software to specify falling edge low level triggered external interrupts.
IE0	TCON.1	Interrupt 0 Edge flag. Set by hardware when external interrupt edge detected. Cleared when interrupt processed.
IT0	TCON.0	Interrupt 0 Type control bit. Set cleared by software to specify falling edge low level triggered external interrupts.

Figure 8. TCON—Timer/Counter Control/Status Register

(MSB)				(LSB)			
SM0	SM1	SM2	REN	TB8	RB8	TI	RI

Symbol	Position	Name and Significance
SM0	SCON.7	Serial port Mode control bit 0. Set/cleared by software (see note).
SM1	SCON.6	Serial port Mode control bit 1. Set/cleared by software (see note).
SM2	SCON.5	Serial port Mode control bit 2. Set by software to disable reception of frames for which bit 8 is zero.
REN	SCON.4	Receiver Enable control bit. Set/cleared by software to enable/disable serial data reception.
TB8	SCON.3	Transmit Bit 8. Set/cleared by hardware to determine state of ninth data bit transmitted in 9-bit UART mode.

Symbol	Position	Name and Significance
RB8	SCON.2	Receive Bit 8. Set/cleared by hardware to indicate state of ninth data bit received.
TI	SCON.1	Transmit Interrupt flag. Set by hardware when byte transmitted. Cleared by software after servicing.
RI	SCON.0	Received Interrupt flag. Set by hardware when byte received. Cleared by software after servicing.

Note— the state of (SM0,SM1) selects:

- (0,0)—Shift register I/O expansion.
- (0,1)—8 bit UART, variable data rate.
- (1,0)—9 bit UART, fixed data rate.
- (1,1)—9 bit UART, variable data rate.

Figure 9. SCON—Serial Port Control/Status Register

These peripheral functions allow special hardware to monitor real-time signal interfacing without bothering the CPU. For example, imagine serial data is arriving from one CRT while being transmitted to another, and one timer/counter is tallying high-speed input transitions while the other measures input pulse widths. During all of this the CPU is thinking about something else.

But how does the CPU know when a reception, transmission, count, or pulse is finished? The 8051 programmer can choose from three approaches.

TCON and SCON contain status bits set by the hardware when a timer overflows or a serial port operation is completed. The first technique reads the control register into the accumulator, tests the appropriate bit, and does a conditional branch based on the result. This "polling" scheme (typically a three-instruction sequence though additional instructions to save and restore the accumulator may sometimes be needed) will surely be familiar to programmers used to multi-chip microcomputer systems and peripheral controller chips. This process is rather cumbersome, especially when monitoring multiple peripherals.

As a second approach, the 8051 can perform a conditional branch based on the state of any control or status bit or input pin in a single instruction; a four instruction sequence could poll the four simultaneous happenings mentioned above in just eight microseconds.

Unfortunately, the CPU must still drop what it's doing to test these bits. A manager cannot do his own work well if he is continuously monitoring his subordinates; they should interrupt him (or her) only when they need attention or guidance. So it is with machines: ideally, the CPU would not have to worry about the peripherals until they require servicing. At that time, it would postpone the

background task long enough to handle the appropriate device, then return to the point where it left off.

This is the basis of the third and generally optimal solution, hardware interrupts. The 8051 has five interrupt sources: one from the serial port when a transmission or reception is complete, two from the timers when overflows occur, and two from input pins INT0 and INT1. Each source may be independently enabled or disabled to allow polling on some sources or at some times, and each may be classified as high or low priority. A high priority source can interrupt a low priority service routine; the manager's boss can interrupt conferences with subordinates. These options are selected by the interrupt enable and priority control registers, IE and IP (Figures 10 and 11).

Each source has a particular program memory address associated with it (Table 3), starting at 0003H (as in the 8048) and continuing at eight-byte intervals. When an event enabled for interrupts occurs the CPU automatically executes an internal subroutine call to the corresponding address. A user subroutine starting at this location (or jumped to from this location) then performs the instructions to service that particular source. After completing the interrupt service routine, execution returns to the background program.

Table 3. 8051 Interrupt Sources and Service Vectors

Interrupt Source	Service Routine Starting Address
(Reset)	0000H
External 0	0003H
Timer/Counter 0	000BH
External 1	0013H
Timer/Counter 1	001BH
Serial Port	0023H

AFN-01502A-15

(MSB)				(LSB)			
EA	—	—	ES	ET1	EX1	ET0	EX0

Symbol	Position	Name and Significance
EA	IE.7	Enable All control bit. Cleared by software to disable all interrupts, independent of the state of IE.4-IE.0.
—	IE.6	(reserved)
—	IE.5	(reserved)
ES	IE.4	Enable Serial port control bit. Set/cleared by software to enable/disable interrupts from TI or RI flags.
ET1	IE.3	Enable Timer 1 control bit. Set/cleared by software to enable/disable interrupts from timer/counter 1.

Symbol	Position	Name and Significance
EX1	IE.2	Enable External interrupt 1 control bit. Set/cleared by software to enable/disable interrupts from INT1.
ET0	IE.1	Enable Timer 0 control bit. Set/cleared by software to enable/disable interrupts from timer/counter 0.
EX0	IE.0	Enable External interrupt 0 control bit. Set/cleared by software to enable/disable interrupts from INT0.

Figure 10. IE—Interrupt Enable Register

(MSB)				(LSB)			
—	—	—	PS	PT1	PX1	PT0	PX0

Symbol	Position	Name and Significance
—	IP.7	(reserved)
—	IP.6	(reserved)
—	IP.5	(reserved)
PS	IP.4	Serial port Priority control bit. Set/cleared by software to specify high/low priority interrupts for Serial port.
PT1	IP.3	Timer 1 Priority control bit. Set/cleared by software to specify high/low priority interrupts for timer/counter 1.

Symbol	Position	Name and Significance
PX1	IP.2	External interrupt 1 Priority control bit. Set/cleared by software to specify high/low priority interrupts for INT1.
PT0	IP.1	Timer 0 Priority control bit. Set/cleared by software to specify high/low priority interrupts for timer/counter 0.
PX0	IP.0	External interrupt 0 Priority control bit. Set/cleared by software to specify high/low priority interrupts for INT0.

Figure 11. IP—Interrupt Priority Control Register

Table 3. 8051 Interrupt Sources and Service Vectors

Interrupt Source	Service Routine Starting Address
(Reset)	0000H
External 0	0003H
Timer/Counter 0	000BH
External 1	0013H
Timer/Counter 1	001BH
Serial Port	0023H

ATM-01502A-10

Table 4. MCS-51™ Instruction Set Description

ARITHMETIC OPERATIONS				DATA TRANSFER (cont.)			
Mnemonic		Description	Byte Cyc	Mnemonic		Description	Byte Cyc
ADD	A,Rn	Add register to Accumulator	1 1	MOVC	A,@A+DPTR	Move Code byte relative to DPTR to A	1 2
ADD	A,direct	Add direct byte to Accumulator	2 1	MOVC	A,@A+PC	Move Code byte relative to PC to A	1 2
ADD	A,@Ri	Add indirect RAM to Accumulator	1 1	MOVX	A,@Ri	Move External RAM (8-bit addr) to A	1 2
ADD	A,#data	Add immediate data to Accumulator	2 1	MOVX	A,@DPTR	Move External RAM (16-bit addr) to A	1 2
ADDC	A,Rn	Add register to Accumulator with Carry	1 1	MOVX	@Ri,A	Move A to External RAM (8-bit addr)	1 2
ADDC	A,direct	Add direct byte to A with Carry flag	2 1	MOVX	@DPTR,A	Move A to External RAM (16-bit addr)	1 2
ADDC	A,@Ri	Add indirect RAM to A with Carry flag	1 1	PUSH	direct	Push direct byte onto stack	2 2
ADDC	A,#data	Add immediate data to A with Carry flag	2 1	POP	direct	Pop direct byte from stack	2 2
SUBB	A,Rn	Subtract register from A with Borrow	1 1	XCH	A,Rn	Exchange register with Accumulator	1 1
SUBB	A,direct	Subtract direct byte from A with Borrow	2 1	XCH	A,direct	Exchange direct byte with Accumulator	2 1
SUBB	A,@Ri	Subtract indirect RAM from A w. Borrow	1 1	XCH	A,@Ri	Exchange indirect RAM with A	1 1
SUBB	A,#data	Subtract immed. data from A w. Borrow	2 1	XCHD	A,@Ri	Exchange low-order Digit ind. RAM w A	1 1
INC	A	Increment Accumulator	1 1	BOOLEAN VARIABLE MANIPULATION			
INC	Rn	Increment register	1 1	Mnemonic		Description	Byte Cyc
INC	direct	Increment direct byte	2 1	CLR	C	Clear Carry flag	1 1
INC	@Ri	Increment indirect RAM	1 1	CLR	bit	Clear direct bit	2 1
DEC	A	Decrement Accumulator	1 1	SETB	C	Set Carry flag	1 1
DEC	Rn	Decrement register	1 1	SETB	bit	Set direct Bit	2 1
DEC	direct	Decrement direct byte	2 1	CPL	C	Complement Carry flag	1 1
DEC	@Ri	Decrement indirect RAM	1 1	CPL	bit	Complement direct bit	2 1
INC DPTR		Increment Data Pointer	1 2	ANL	C,bit	AND direct bit to Carry flag	2 2
MUL	AB	Multiply A & B	1 4	ANL	C, bit	AND complement of direct bit to Carry	2 2
DIV	AB	Divide A by B	1 4	ORL	C,bit	OR direct bit to Carry flag	2 2
DA	A	Decimal Adjust Accumulator	1 1	ORL	C, bit	OR complement of direct bit to Carry	2 2
				MOV	C,bit	Move direct bit to Carry flag	2 2
				MOV	bit,C	Move Carry flag to direct bit	2 2
LOGICAL OPERATIONS				PROGRAM AND MACHINE CONTROL			
Mnemonic		Destination	Byte Cyc	Mnemonic		Description	Byte Cyc
ANL	A,Rn	AND register to Accumulator	1 1	ACALL	addr11	Absolute Subroutine Call	2 2
ANL	A,direct	AND direct byte to Accumulator	2 1	LCALL	addr16	Long Subroutine Call	3 2
ANL	A,@Ri	AND indirect RAM to Accumulator	1 1	RET		Return from subroutine	1 2
ANL	A,#data	AND immediate data to Accumulator	2 1	RETI		Return from interrupt	1 2
ANL	direct,A	AND Accumulator to direct byte	2 1	AJMP	addr11	Absolute Jump	2 2
ANL	direct,#data	AND immediate data to direct byte	3 2	LJMP	addr16	Long Jump	3 2
ORL	A,Rn	OR register to Accumulator	1 1	SJMP	rel	Short Jump (relative addr)	2 2
ORL	A,direct	OR direct byte to Accumulator	2 1	JMP	@A+DPTR	Jump indirect relative to the DPTR	1 1
ORL	A,@Ri	OR indirect RAM to Accumulator	1 1	JZ	rel	Jump if Accumulator is Zero	2 2
ORL	A,#data	OR immediate data to Accumulator	2 1	JC	rel	Jump if Accumulator is Not Zero	2 2
ORL	direct,A	OR Accumulator to direct byte	2 1	JNC	rel	Jump if Carry flag is set	2 2
ORL	direct,#data	OR immediate data to direct byte	3 2	JB	bit,rel	Jump if No Carry flag	2 2
XRL	A,Rn	Exclusive-OR register to Accumulator	1 1	JNB	bit,rel	Jump if direct Bit set	3 2
XRL	A,direct	Exclusive-OR direct byte to Accumulator	2 1	JNB	bit,rel	Jump if direct Bit Not set	3 2
XRL	A,@Ri	Exclusive-OR indirect RAM to A	1 1	JBC	bit,rel	Jump if direct Bit is set & Clear bit	3 2
XRL	A,#data	Exclusive-OR immediate data to A	2 1	CJNE	A,direct,rel	Compare direct to A & Jump if Not Equal	3 2
XRL	direct,A	Exclusive-OR Accumulator to direct byte	2 1	CJNE	A,#data,rel	Comp. immed. to A & Jump if Not Equal	3 2
XRL	direct,#data	Exclusive-OR immediate data to direct	3 2	CJNE	Rn,#data,rel	Comp. immed. to reg. & Jump if Not Equal	3 2
CLR	A	Clear Accumulator	1 1	CJNE	@Ri,#data,rel	Comp. immed. to ind. & Jump if Not Equal	3 2
CPL	A	Complement Accumulator	1 1	DJNZ	Rn,rel	Decrement register & Jump if Not Zero	2 2
RL	A	Rotate Accumulator Left	1 1	DJNZ	direct,rel	Decrement direct & Jump if Not Zero	3 2
RLC	A	Rotate A Left through the Carry flag	1 1	NOP		No operation	1 1
RR	A	Rotate Accumulator Right	1 1				
RRC	A	Rotate A Right through Carry flag	1 1				
SWAP	A	Swap nibbles within the Accumulator	1 1				
DATA TRANSFER				Notes on data addressing modes:			
Mnemonic		Description	Byte Cyc	Rn		Working register R0-R7	
MOV	A,Rn	Move register to Accumulator	1 1	direct		128 internal RAM locations, any I/O port, control or status register	
MOV	A,direct	Move direct byte to Accumulator	2 1	@Ri		Indirect internal RAM location addressed by register R0 or R1	
MOV	A,@Ri	Move indirect RAM to Accumulator	1 1	#data		8-bit constant included in instruction	
MOV	A,#data	Move immediate data to Accumulator	2 1	#data16		16-bit constant included as bytes 2 & 3 of instruction	
MOV	Rn,A	Move Accumulator to register	1 1	bit		128 software flags, any I/O pin, control or status bit	
MOV	Rn,direct	Move direct byte to register	2 2	Notes on program addressing modes:			
MOV	Rn,direct,A	Move Accumulator to direct byte	2 1	addr16		Destination address for LCALL & LJMP may be anywhere within the 64-Kilobyte program memory address space	
MOV	direct,Rn	Move register to direct byte	2 2	addr11		Destination address for ACALL & AJMP will be within the same 2-Kilobyte page of program memory as the first byte of the following instruction	
MOV	direct,direct	Move direct byte to direct	3 2	rel		SJMP and all conditional jumps include an 8-bit offset byte. Range is +127 -128 bytes relative to first byte of the following instruction	
MOV	direct,@Ri	Move indirect RAM to direct byte	2 2				
MOV	direct,#data	Move immediate data to direct byte	3 2				
MOV	@Ri,A	Move Accumulator to indirect RAM	1 1				
MOV	@Ri,direct	Move direct byte to indirect RAM	2 2				
MOV	@Ri,#data	Move immediate data to indirect RAM	2 2				
MOV	DPTR,#data16	Load Data Pointer with a 16-bit constant	3 2				
				All mnemonics copyrighted © Intel Corporation 1979			

All mnemonics copyrighted © Intel Corporation 1979

3. INSTRUCTION SET AND ADDRESSING MODES

The 8051 instruction set is extremely regular, in the sense that most instructions can operate with variables from several different physical or logical address spaces. Before getting deeply enmeshed in the instruction set proper, it is important to understand the details of the most common data addressing modes. Whereas Table 4 summarizes the instructions set broken down by functional

group, this chapter starts with the addressing mode classes and builds to include the related instructions.

Data Addressing Modes

MCS-51 assembly language instructions consist of an operation mnemonic and zero to three operands separated by commas. In two operand instructions the destination is specified first, then the source. Many byte-wide data

operations (such as ADD or MOV) inherently use the accumulator as a source operand and/or to receive the result. For the sake of clarity the letter "A" is specified in the source or destination field in all such instructions. For example, the instruction,

```
ADD A,<source>
```

will add the variable <source> to the accumulator, leaving the sum in the accumulator.

The operand designated "<source>" above may use any of four common logical addressing modes:

- Register—one of the working registers in the currently enabled bank.
- Direct—an internal RAM location, I/O port, or special-function register.
- Register-indirect—an internal RAM location, pointed to by a working register.
- Immediate data—an eight-bit constant incorporated into the instruction.

The first three modes provide access to the internal RAM and Hardware Register address spaces, and may therefore be used as source or destination operands; the last mode accesses program memory and may be a source operand only.

(It is hard to show a "typical application" of any instruction without involving instructions not yet described. The following descriptions use only the self-explanatory ADD and MOV instructions to demonstrate how the four addressing modes are specified and used. Subsequent examples will become increasingly complex.)

Register Addressing

The 8051 programmer has access to eight "working registers," numbered R0-R7. The least-significant three-bits of the instruction opcode indicate one register within this logical address space. Thus, a function code and operand address can be combined to form a short (one byte) instruction (Figure 12.a).

The 8051 assembly language indicates register addressing with the symbol Rn (where n is from 0 to 7) or with a symbolic name previously defined as a register by the EQUate or SET directives. (For more information on assembler directives see the Macro Assembler Reference Manual.)

Example 1—Adding Two Registers Together

```
REGADR ADD CONTENTS OF REGISTER 0
        TO CONTENTS OF REGISTER 0

REGADR MOV A,R0
ADD A,R1
MOV R0,A
```

There are four such banks of working registers, only one of which is active at a time. Physically, they occupy the first 32 bytes of on-chip data RAM (addresses 0-1FH). PSW bits 4 and 3 determine which bank is active. A

hardware reset enables register bank 0; to select a different bank the programmer modifies PSW bits 4 and 3 accordingly.

Example 2—Selecting Alternate Memory Banks

```
MOV PSW,#00010000B SELECT BANK 2
```

Register addressing in the 8051 is the same as in the 8048 family, with two enhancements: there are four banks rather than one or two, and 16 instructions (rather than 12) can access them.

Direct Byte Addressing

Direct addressing can access any on-chip variable or hardware register. An additional byte appended to the opcode specifies the location to be used (Figure 12.b).

Depending on the highest order bit of the direct address byte, one of two physical memory spaces is selected. When the direct address is between 0 and 127 (00H-7FH) one of the 128 low-order on-chip RAM locations is used. (Future microcomputers based on the MCS-51™ architecture may incorporate more than 128 bytes of on-chip RAM. Even if this is the case, only the low-order 128 bytes will be directly addressable. The remainder would be accessed indirectly or via the stack pointer.)

Example 3—Adding RAM Location Contents

```
DIRADR ADD CONTENTS OF RAM LOCATION 41H
        TO CONTENTS OF RAM LOCATION 40H

DIRADR MOV A,40H
ADD A,41H
MOV 40H,A
```

All I/O ports and special function, control, or status registers are assigned addresses between 128 and 255 (80H-0FFH). When the direct address byte is between these limits the corresponding hardware register is accessed. For example, Ports 0 and 1 are assigned direct addresses 80H and 90H, respectively. A complete list is presented in Table 5. Don't waste your time trying to memorize the addresses in Table 5. Since programs using absolute addresses for function registers would be difficult to write or understand, ASM51 allows and understands the abbreviations listed instead.

Example 4—Adding Input Port Data to Output Port Data

```
PRTADR ADD DATA INPUT ON PORT 1
        TO DATA PREVIOUSLY OUTPUT
        ON PORT 0

PRTADR MOV A,PO
ADD A,PI
MOV PO,A
```

Direct addressing allows all special-function registers in the 8051 to be read, written, or used as instruction operands. In general, this is the *only* method used for accessing I/O ports and special-function registers. If direct addressing is used with special-function register addresses other than those listed, the result of the instruction is undefined.

The 8048 does not have or need any generalized direct addressing mode, since there are only five special registers (BUS, P1, P2, PSW, & T) rather than twenty. Instead, 16 special 8048 opcodes control output bits or read or write each register to the accumulator. These functions are all subsumed by four of the 27 direct addressing instructions of the 8051.

Table 5. 8051 Hardware Register Direct Addresses

Register	Address	Function
P0	80H*	Port 0
SP	81H	Stack Pointer
DPL	82H	Data Pointer (Low)
DPH	83H	Data Pointer (High)
TCN	88H*	Timer register
TMOD	89H	Timer Mode register
TL0	8AH	Timer 0 Low byte
TL1	8BH	Timer 1 Low byte
TH0	8CH	Timer 0 High byte
TH1	8DH	Timer 1 High byte
P1	90H*	Port 1
SCON	98H*	Serial Port Control register
SBUF	99H	Serial Port data Buffer
P2	0A0H*	Port 2
IE	0A8H*	Interrupt Enable register
P3	0B0H*	Port 3
IP	0B8H*	Interrupt Priority register
PSW	0D0H*	Program Status Word
ACC	0E0H*	Accumulator (direct address)
B	0F0H*	B register

* = bit addressable register.

Register-Indirect Addressing

How can you handle variables whose locations in RAM are determined, computed, or modified while the program is running? This situation arises when manipulating sequential memory locations, indexed entries within tables in RAM, and multiple precision or string operations. Register or Direct addressing cannot be used, since their operand addresses are fixed at assembly time.

The 8051 solution is "register-indirect RAM addressing." R0 and R1 of each register bank may operate as index or pointer registers, their contents indicating an address into RAM. The internal RAM location so addressed is the actual operand used. The least significant bit of the instruction opcode determines which register is used as the "pointer" (Figure 12.c).

In the 8051 assembly language, register-indirect addressing is represented by a commercial "at" sign ("@") preceding R0, R1, or a symbol defined by the user to be equal to R0 or R1.

Example 5—Indirect Addressing

```

INDADR ADD CONTENTS OF MEMORY LOCATION
        ADDRESSED BY REGISTER 1
        TO CONTENTS OF RAM LOCATION
        ADDRESSED BY REGISTER 0
INDADR MOV A, @R0
ADD     A, @R1
MOV     @R0, A

```

Indirect addressing on the 8051 is the same as in the 8048 family, except that all eight bits of the pointer register contents are significant; if the contents point to a non-existent memory location (i.e., an address greater than 7FH on the 8051) the result of the instruction is undefined. (Future microcomputers based on the MCS-51™ architecture could implement additional memory in the on-chip RAM logical address space at locations above 7FH.) The 8051 uses register-indirect addressing for five new instructions plus the 13 on the 8048.

Immediate Addressing

When a source operand is a constant rather than a variable (i.e., the instruction uses a value known at assembly time), then the constant can be incorporated into the instruction. An additional instruction byte specifies the value used (Figure 12.d).

The value used is fixed at the time of ROM manufacture or EPROM programming and may not be altered during program execution. In the assembly language immediate operands are preceded by a number sign ("#"). The operand may be either a numeric string, a symbolic variable, or an arithmetic expression using constants.

Example 6—Adding Constants Using Immediate Addressing

```

IMMADR ADD THE CONSTANT 12 (DECIMAL)
        TO THE CONSTANT 34 (DECIMAL)
        LEAVE SUM IN ACCUMULATOR
IMMADR MOV A, #12
ADD     A, #34

```

The preceding example was included for consistency; it has little practical value. Instead, ASM51 could compute the sum of two constants at assembly time.

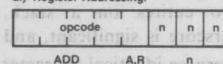
Example 7—Adding Constants Using ASM51 Capabilities

```

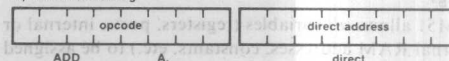
ASM51M LOAD ACC WITH THE SUM OF
        THE CONSTANT 12 (DECIMAL) AND
        THE CONSTANT 34 (DECIMAL)
ASM51M MOV A, # (12+34)

```

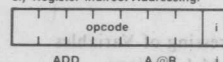
a.) Register Addressing:



b.) Direct Addressing:



c.) Register-Indirect Addressing:



d.) Immediate Addressing:

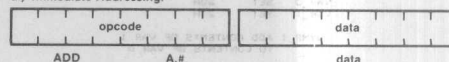


Figure 12. Data Addressing Machine Code Formats

AFN-01502A-19

Addressing Mode Combinations

The above examples all demonstrated the use of the four data-addressing modes in two-operand instructions (MOV, ADD) which use the accumulator as one operand. The operations ADDC, SUBB, ANL, ORL, and XRL (all to be discussed later) could be substituted for ADD in each example. The first three modes may be also be used for the XCH operation or, in combination with the Immediate Addressing mode (and an additional byte), loaded with a constant. The one-operand instructions INC and DEC, DJNZ, and CJNE may all operate on the accumulator, or may specify the Register, Direct, and Register-indirect addressing modes. Exception: as in the 8048, DJNZ cannot use the accumulator or indirect addressing. (The PUSH and POP operations cannot inherently address the accumulator as a special register either. However, all three can *directly* address the accumulator as one of the twenty special-function registers by putting the symbol "ACC" in the operand field.)

Advantages of Symbolic Addressing

Like most assembly or higher-level programming languages, ASM51 allows instructions or variables to be given appropriate, user-defined symbolic names. This is done for instruction lines by putting a label followed by a colon (":") before the instruction proper, as in the above examples. Such symbols must start with an alphabetic character (remember what distinguished BACH from 0BACH?), and may include any combination of letters, numbers, question marks ("?",) and underscores ("_"). For very long names only the first 31 characters are relevant.

Assembly language programs may intermix upper- and lower-case letters arbitrarily, but ASM51 converts both to upper-case. For example, ASM51 will internally process an "I" for an "i" and, of course, "A_TOOTH" for "a_tooth."

The underscore character makes symbols easier to read and can eliminate potential ambiguity (as in the label for a subroutine to switch two entires on a stack, "S_EXCHANGE"). The underscore is significant, and would distinguish between otherwise-identical character strings.

ASM51 allows *all* variables (registers, ports, internal or external RAM addresses, constants, etc.) to be assigned labels according to these rules with the EQUate or SET directives.

Example 8—Symbolic Addressing of Variables Defined as RAM Locations

VAR_0	SET	20H
VAR_1	SET	21H
SYMB_1 ADD CONTENTS OF VAR_1		
TO CONTENTS OF VAR_0		
SYMB_1	MOV	A, VAR_0
	ADD	A, VAR_1
	MOV	VAR_0, A

Notice from Table 4 that the MCS-51™ instruction set has relatively few instruction mnemonics (abbreviations) for the programmer to memorize. Different data types or addressing modes are determined by the operands specified, rather than variations on the mnemonic. For example, the mnemonic "MOV" is used by 18 different instructions to operate on three data types (bit, byte, and address). The fifteen versions which move byte variables between the logical address spaces are diagrammed in Figure 13. Each arrow shows the direction of transfer from source to destination.

Notice also that for most instructions allowing register addressing there is a corresponding direct addressing instruction and vice versa. This lets the programmer begin writing 8051 programs as if (s)he has access to 128 different registers. When the program has evolved to the point where the programmer has a fairly accurate idea how often each variable is used, he/she may allocate the working registers in each bank to the most "popular" variables. (The assembly cross-reference option will show exactly how often and where each symbol is referenced.) If symbolic addressing is used in writing the source program only the lines containing the symbol definition will need to be changed; the assembler will produce the appropriate instructions even though the rest of the program is left untouched. Editing only the first two lines of Example 8 will shrink the six-byte code segment produced in half.

How are instruction sets "counted"? There is no standard practice; different people assessing the same CPU using different conventions may arrive at different totals.

Each operation is then broken down according to the different addressing modes (or combinations of addressing modes) it can accommodate. The "CLR" mnemonic is used by two instructions with respect to bit variables ("CLR C" and "CLR bit") and once ("CLR A") with regards to bytes. This expansion yields the 111 separate instructions of Table 4.

The method used for the MCS-51 instruction set first breaks it down into "operations": a basic function applied to a single data type. For example, the four versions of the ADD instruction are grouped to form one operation — addition of eight-bit variables. The six forms of the ANL instruction for byte variables make up a different operation; the two forms of ANL which operate on *bits* are considered still another. The MOV mnemonic is used by three different operation classes, depending on whether bit, byte, or 16-bit values are affected. Using this terminology the 8051 can perform 51 different operations.

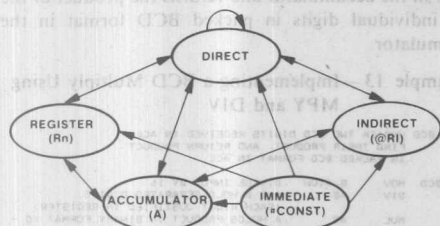


Figure 13. Road map for moving data bytes

Example 9—Redeclaring Example 8 Symbols as Registers

```

VAR_0 SET RO
VAR_1 SET R1
SYMB_2 ADD CONTENTS OF VAR_1
        TO CONTENTS OF VAR_0
SYMB_2 MOV A, VAR_0
        ADD A, VAR_1
        MOV VAR_0, A
    
```

Arithmetic Instruction Usage — ADD, ADDC, SUBB and DA

The ADD instruction adds a byte variable with the accumulator, leaving the result in the accumulator. The carry flag is set if there is an overflow from bit 7 and cleared otherwise. The AC flag is set to the carry-out from bit 3 for use by the DA instruction described later. ADDC adds the previous contents of the carry flag with the two byte variables, but otherwise is the same as ADD.

The SUBB (subtract with borrow) instruction subtracts the byte variable indicated and the contents of the carry flag together from the accumulator, and puts the result back in the accumulator. The carry flag serves as a "Borrow Required" flag during subtraction operations; when a greater value is subtracted from a lesser value (as in subtracting 5 from 1) requiring a borrow into the highest order bit, the carry flag is set; otherwise it is cleared.

When performing signed binary arithmetic, certain combinations of input variables can produce results which seem to violate the Laws of Mathematics. For example, adding 7FH (127) to itself produces a sum of 0FEH, which is the two's complement representation of -2 (refer back to Table 2). In "normal" arithmetic, two positive values can't have a negative sum. Similarly, it is normally impossible to subtract a positive value from a negative value and leave a positive result — but in two's complement there are instances where this too may happen. Fundamentally, such anomalies occur when the magnitude of the resulting value is too great to "fit" into the seven bits allowed for it; there is no one-byte two's complement representation for 254, the true sum of 127 and 127.

The MCS-51™ processors detect whether these situations occur and indicate such errors with the OV flag. (OV may be tested with the conditional jump instructions JB and JNB, described under the Boolean Processor chapter.)

At a hardware level, OV is set if there is a carry out of bit 6 but not out of bit 7, or a carry out of bit 7 but not out of bit 6. When adding signed integers this indicates a negative number produced as the sum of two positive operands, or a positive sum from two negative operands; on SUBB this indicates a negative result after subtracting a negative number from a positive number, or a positive result when a positive number is subtracted from a negative number.

The ADDC and SUBB instructions incorporate the previous state of the carry (borrow) flag to allow multiple precision calculations by repeating the operation with successively higher-order operand bytes. In either case, the carry must be cleared before the first iteration.

If the input data for a multiple precision operation is an unsigned string of integers, upon completion the carry flag will be set if an overflow (for ADDC) or underflow (for SUBB) occurs. With two's complement signed data (i.e., if the most significant bit of the original input data indicates the sign of the string), the overflow flag will be set if overflow or underflow occurred.

Example 10—String Subtraction with Signed Overflow Detection

```

SUBSTR SUBTRACT STRING INDICATED BY R1
        FROM STRING INDICATED BY R0 TO
        PRECISION INDICATED BY R2
        CHECK FOR SIGNED UNDERFLOW WHEN DONE

SUBSTR CLR C          BORROW = 0
SUBS1  MOV A, R0
SUBB   A, R1          SUBTRACT NEXT PLACE
MOV    R0, A
INC    R0             BUMP POINTERS
INC    R1
DJNZ   R2, SUBS1      LOOP AS NEEDED
                        WHEN DONE, TEST IF OVERFLOW OCCURRED
                        ON LAST ITERATION OF LOOP
UNB    OV, OV_OK      (OVERFLOW RECOVERY ROUTINE)
OV_OK  RET            RETURN
    
```

Decimal addition is possible by using the DA instruction in conjunction with ADD and/or ADDC. The eight-bit binary value in the accumulator resulting from an earlier addition of two variables (each a packed BCD digit-pair) is adjusted to form two BCD digits of four bits each. If the contents of accumulator bits 3-0 are greater than nine (xxxx1010-xxxx1111), or if the AC flag had been set, six is added to the accumulator producing the proper BCD digit in the low-order nibble. (This addition might itself set the carry flag, but would not clear the carry flag.) If the carry flag is set, or if the four high-order bits now exceed nine (1010xxxx-1111xxxx), these bits are incremented by six. The carry flag is left set if originally set or if either addition of six produces a carry out of the highest-order bit, indicating the sum of the original two BCD variables is greater than or equal to decimal 100.

Example 11—Two Byte Decimal Add with Registers and Constants

```
BCDADD ADD THE CONSTANT 1,234 (DECIMAL) TO THE
        CONTENTS OF REGISTER PAIR (R3) (R2)
        (ALREADY A 4 BCD-DIGIT VARIABLE)

BCDADD MOV A, R2
ADD     A, #34H
DA      A
MOV     R2, A
MOV     A, R3
ADDC    A, #12H
DA      A
MOV     R3, A
RET
```

Multiplication and Division

The instruction "MUL AB" multiplies the unsigned eight-bit integer values held in the accumulator and B-registers. The low-order byte of the sixteen-bit product is left in the accumulator, the higher-order byte in B. If the high-order eight-bits of the product are all zero the overflow flag is cleared; otherwise it is set. The programmer can poll OV to determine when the B register is non-zero and must be processed.

"DIV AB" divides the unsigned eight-bit integer in the accumulator by the unsigned eight-bit integer in the B-register. The integer part of the quotient is returned in the accumulator; the remainder in the B-register. If the B-register originally contained 00H then the overflow flag will be set to indicate a division error, and the values returned will be undefined. Otherwise OV is cleared.

The divide instruction is also useful for purposes such as radix conversion or separating bit fields of the accumulator. A short subroutine can convert an eight-bit unsigned binary integer in the accumulator (between 0 & 255) to a three-digit (two byte) BCD representation. The hundred's digit is returned in one register (HUND) and the ten's and one's digits returned as packed BCD in another (TENONE).

Example 12—Use of DIV Instruction for Radix Conversion

```
;BINBCD CONVERT 8-BIT BINARY VARIABLE IN ACC
; TO 3-DIGIT PACKED BCD FORMAT
; HUNDREDS' PLACE LEFT IN VARIABLE 'HUND',
; TENS' AND ONES' PLACES IN 'TENONE'

HUND EQU 21H
TENONE EQU 22H

BINBCD MOV A, #100 ;DIVIDE BY 100 TO
DIV AB ;DETERMINE NUMBER OF HUNDREDS
MOV HUND, A
MOV A, #10 ;DIVIDE REMAINDER BY 10 TO
XCH A, B ;DETERMINE # OF TENS LEFT
DIV AB ;TENS DIGIT IN ACC, REMAINDER IS ONES
;DIGIT
SWAP A
ADD A, B ;PACK BCD DIGITS IN ACC
MOV TENONE, A
RET
```

The divide instruction can also separate eight bits of data in the accumulator into sub-fields. For example, packed BCD data may be separated into two nibbles by dividing the data by 16, leaving the high-nibble in the accumulator and the low-order nibble (remainder) in B. The two digits may then be operated on individually or in conjunction with each other. This example receives two packed BCD

digits in the accumulator and returns the product of the two individual digits in packed BCD format in the accumulator.

Example 13—Implementing a BCD Multiply Using MPY and DIV

```
MULBCD UNPACK TWO BCD DIGITS RECEIVED IN ACC,
        FIND THEIR PRODUCT, AND RETURN PRODUCT
        IN PACKED BCD FORMAT IN ACC

MULBCD MOV B, #10H ;DIVIDE INPUT BY 16
DIV AB ;A & B HOLD SEPARATED DIGITS
        (EACH RIGHT JUSTIFIED IN REGISTER)
MUL AB ;A HOLDS PRODUCT IN BINARY FORMAT (0 -
        99(DECIMAL) = 0 - 63H)
MOV B, #10 ;DIVIDE PRODUCT BY 10
DIV AB ;A HOLDS # OF TENS, B HOLDS REMAINDER
SWAP A
ORL A, B ;PACK DIGITS
RET
```

Logical Byte Operations — ANL, ORL, XRL

The instructions ANL, ORL, and XRL perform the logical functions AND, OR, and/or Exclusive-OR on the two byte variables indicated, leaving the results in the first. No flags are affected. (A word to the wise — do not vocalize the first two mnemonics in mixed company.)

These operations may use all the same addressing modes as the arithmetics (ADD, etc.) but unlike the arithmetics, they are not restricted to operating on the accumulator. Directly addressed bytes may be used as the destination with either the accumulator or a constant as the source. These instructions are useful for clearing (ANL), setting (ORL), or complementing (XRL) one or more bits in a RAM, output ports, or control registers. The pattern of bits to be affected is indicated by a suitable mask byte. Use immediate addressing when the pattern to be affected is known at assembly time (Figure 14); use the accumulator versions when the pattern is computed at run-time.

I/O ports are often used for parallel data in formats other than simple eight-bit bytes. For example, the low-order five bits of port 1 may output an alphabetic character code (hopefully) without disturbing bits 7-5. This can be a simple two-step process. First, clear the low-order five pins with an ANL instruction; then set those pins corresponding to ones in the accumulator. (This example assumes the three high-order bits of the accumulator are originally zero.)

Example 14—Reconfiguring Port Size with Logical Byte Instructions

```
OUT_PX ANL P1, #11100000B ;CLEAR BITS P1.4 - P1.0
ORL P1, A ;SET P1 PINS CORRESPONDING TO SET ACC
RET
```

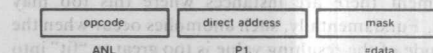


Figure 14. Instruction Pattern for Logical Operation Special Addressing Modes

In this example, low-order bits remaining high may "glitch" low for one machine cycle. If this is undesirable, use a slightly different approach. First, set all pins corresponding to accumulator one bits, then clear the pins corresponding to zeroes in low-order accumulator bits. Not all bits will change from original to final state at the same instant, but no bit makes an intermediate transition.

Example 15 — Reconfiguring I/O Port Size without Glitching

```
ALT_PX ORL P1,A
        ORL A,#11100000B
        ANL RET P1,A
```

Program Control — Jumps, Calls, Returns

Whereas the 8048 only has a single form of the simple jump instruction, the 8051 has three. Each causes the program to unconditionally jump to some other address. They differ in how the machine code represents the destination address.

LJMP (Long Jump) encodes a sixteen-bit address in the second and third instruction bytes (Figure 15.a); the destination may be anywhere in the 64 Kilobyte program memory address space.

The two-byte AJMP (Absolute Jump) instruction encodes its destination using the same format as the 8048: address bits 10 through 8 form a three bit field in the opcode and address bits 7 through 0 form the second byte (Figure 15.b). Address bits 15-12 are unchanged from the (incremented) contents of the P.C., so AJMP can only be used when the destination is known to be within the same 2K memory block. (Otherwise ASM51 will point out the error.)

A different two-byte jump instruction is legal with any nearby destination, regardless of memory block boundaries or "pages." SJMP (Short Jump) encodes the destination with a program counter-relative address in the second byte (Figure 15.c). The CPU calculates the

destination at run-time by adding the signed eight-bit displacement value to the incremented P.C. Negative offset values will cause jumps up to 128 bytes backwards; positive values up to 127 bytes forwards. (SJMP with 00H in the machine code offset byte will proceed with the following instruction).

In keeping with the 8051 assembly language goal of minimizing the number of instruction mnemonics, there is a "generic" form of the three jump instructions. ASM51 recognizes the mnemonic JMP as a "pseudo-instruction," translating it into the machine instructions LJMP, AJMP, or SJMP, depending on the destination address.

Like SJMP, all conditional jump instructions use relative addressing. JZ (Jump if Zero) and JNZ (Jump if Not Zero) monitor the state of the accumulator as implied by their names, while JC (Jump on Carry) and JNC (Jump on No Carry) test whether or not the carry flag is set. All four are two-byte instructions, with the same format as Figure 15.c. JB (Jump on Bit), JNB (Jump on No Bit) and JBC (Jump on Bit then Clear Bit) can test any status bit or input pin with a three byte instruction; the second byte specifies which bit to test and the third gives the relative offset value.

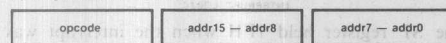
There are two subroutine-call instructions, LCALL (Long Call) and ACALL (Absolute Call). Each increments the P.C. to the first byte of the following instruction, then pushes it onto the stack (low byte first). Saving both bytes increments the stack pointer by two. The subroutine's starting address is encoded in the same ways as LJMP and AJMP. The generic form of the call operation is the mnemonic CALL, which ASM51 will translate into LCALL or ACALL as appropriate.

The return instruction RET pops the high- and low-order bytes of the program counter successively from the stack, decrementing the stack pointer by two. Program execution continues at the address previously pushed: the first byte of the instruction immediately following the call.

When an interrupt request is recognized by the 8051 hardware, two things happen. Program control is automatically "vectored" to one of the interrupt service routine starting addresses by, in effect, forcing the CPU to process an LCALL instead of the next instruction. This automatically stores the return address on the stack. (Unlike the 8048, no status information is automatically saved.)

Secondly, the interrupt logic is disabled from accepting any other interrupts from the same or lower priority. After completing the interrupt service routine, executing an RETI (Return from Interrupt) instruction will return execution to the point where the background program was interrupted — just like RET — while restoring the interrupt logic to its previous state.

a.) Long Jump (LJMP addr16):



b.) Absolute Jump (AJMP addr11):



c.) Short Jump (SJMP rel):



Figure 15. Jump Instruction Machine Code Formats

Operate-and-branch instructions — CJNE, DJNZ

Two groups of instructions combine a byte operation with a conditional jump based on the results.

CJNE (Compare and Jump if Not Equal) compares two byte operands and executes a jump if they disagree. The carry flag is set following the rules for subtraction: if the unsigned integer value of the first operand is less than that of the second it is set; otherwise, it is cleared. However, neither operand is modified.

The CJNE instruction provides, in effect, a one-instruction "case" statement. This instruction may be executed repeatedly, comparing the code variable to a list of "special case" value: the code segment following the instruction (up to the destination label) will be executed only if the operands match. Comparing the accumulator or a register to a series of constants is a convenient way to check for special handling or error conditions; if none of the cases match the program will continue with "normal" processing.

A typical example might be a word processing device which receives ASCII characters through the serial port and drives a thermal hard-copy printer. A standard routine translates "printing" characters to bit patterns, but control characters (<CR> <LF> <BEL> <ESC> or <SP>) must invoke corresponding special routines. Any other character with an ASCII code less than 20H should be translated into the <NUL> value, 00H, and processed with the printing characters.

Example 16 — Case Statements Using CJNE

```
CHAR EQU R7 ; CHARACTER CODE VARIABLE
INTP EQU CHAR, #7FH, INTP_1 ; (SPECIAL ROUTINE FOR RUBOUT CODE)
INTP_1 CJNE CHAR, #07H, INTP_2 ; (SPECIAL ROUTINE FOR BELL CODE)
INTP_2 CJNE CHAR, #0AH, INTP_3 ; (SPECIAL ROUTINE FOR LFEEED CODE)
INTP_3 CJNE CHAR, #0DH, INTP_4 ; (SPECIAL ROUTINE FOR RETURN CODE)
INTP_4 CJNE CHAR, #1BH, INTP_5 ; (SPECIAL ROUTINE FOR ESCAPE CODE)
INTP_5 CJNE CHAR, #20H, INTP_6 ; (SPECIAL ROUTINE FOR SPACE CODE)
INTP_6 JC PRINTC ; JUMP IF CODE > 20H
MOV CHAR, #0 ; REPLACE CONTROL CHARACTERS WITH NULL CODE
PRINTC ; PROCESS STANDARD PRINTING CHARACTER
RET
```

DJNZ (Decrement and Jump if Not Zero) decrements the register or direct address indicated and jumps if the result is not zero, without affecting any flags. This provides a simple means for executing a program loop a given number of times, or for adding a moderate time delay (from 2 to 512 machine cycles) with a single instruction. For example, a 99-usec. software delay loop can be added to code forcing an I/O pin low with only two instructions.

Example 17 — Inserting a Software Delay with DJNZ

```
CLR WR ; WR
MOV R2, #99 ; DJNZ R2, $
DJNZ R2, $ ; SETB WR ; WR
```

The dollar sign in this example is a special character meaning "the address of this instruction." It is useful in eliminating instruction labels on the same or adjacent source lines. CJNE and DJNZ (like all conditional jumps) use program-counter relative addressing for the destination address.

Stack Operations — PUSH, POP

The PUSH instruction increments the stack pointer by one, then transfers the contents of the single byte variable indicated (direct addressing only) into the internal RAM location addressed by the stack pointer. Conversely, POP copies the contents of the internal RAM location addressed by the stack pointer to the byte variable indicated, then decrements the stack pointer by one.

(Stack Addressing follows the same rules, and addresses the same locations as Register-indirect. Future microcomputers based on the MCS-51™ CPU could have up to 256 bytes of RAM for the stack.)

Interrupt service routines must not change any variable or hardware registers modified by the main program, or else the program may not resume correctly. (Such a change might look like a spontaneous random error.) Resources used or altered by the service routine (Accumulator, PSW, etc.) must be saved and restored to their previous value before returning from the service routine. PUSH and POP provide an efficient and convenient way to save register states on the stack.

Example 18 — Use of the Stack for Status Saving on Interrupts

```
LOC_TMP EQU 0 ; REMEMBER LOCATION COUNTER
ORG 0003H ; STARTING ADDRESS FOR INTERRUPT ROUTINE
LUMP ; JUMP TO ACTUAL SERVICE ROUTINE, LOCATED ELSEWHERE
ORG 0004H ; RESTORE LOCATION COUNTER
SERVER PUSH PSW ; SAVE ACCUMULATOR (NOTE DIRECT ADDRESSING NOTATION)
PUSH B ; SAVE B REGISTER
PUSH DPH ; SAVE DATA POINTER
MOV PSW, #00001000B ; SELECT REGISTER BANK 1
POP DPH ; RESTORE REGISTERS IN REVERSE ORDER
POP DPL
POP B
POP ACC
POP PSW ; RESTORE PSW AND RE-SELECT ORIGINAL REGISTER BANK
RETI ; RETURN TO MAIN PROGRAM AND RESTORE INTERRUPT LOGIC
```

If the SP register held 1FH when the interrupt was detected, then while the service routine was in progress the stack would hold the registers shown in Figure 16; SP would contain 26H.

The example shows the most general situation; if the service routine doesn't alter the B-register and data pointer, for example, the instructions saving and restoring those registers would not be necessary.

The stack may also pass parameters to and from subroutines. The subroutine can indirectly address the parameters derived from the contents of the stack pointer.

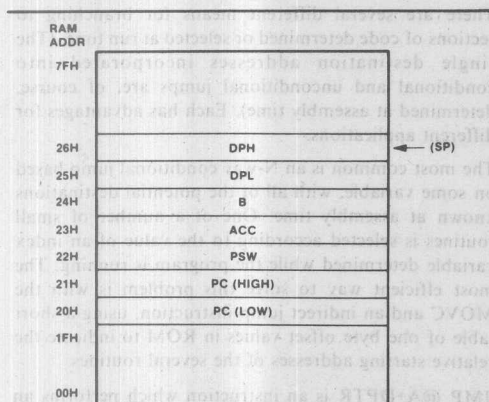


Figure 16. Stack contents during interrupt

One advantage here is simplicity. Variables need not be allocated for specific parameters, a potentially large number of parameters may be passed, and different calling programs may use different techniques for determining or handling the variables.

For example, the following subroutine reads out a parameter stored on the stack by the calling program, uses the low order bits to access a local look-up table holding bit patterns for driving the coils of a four phase stepper motor, and stores the appropriate bit pattern back in the same position on the stack before returning. The accumulator contents are left unchanged.

Example 19—Passing Variable Parameters to Subroutines Using the Stack

```

NXTPOS MOV RO, SP      ; ACCESS LOCATION PARAMETER PUSHED INTO
DEC RO      ;
DEC RO      ;
XCH A, @RO   ; READ INPUT PARAMETER AND SAVE
            ; ACCUMULATOR
ANL A, #03H  ; MASK ALL BUT LOW-ORDER TWO BITS
ADD A, #2    ; ALLOW FOR OFFSET FROM MOVX TO TABLE
MOVC A, @A+PC ; READ LOOK-UP TABLE ENTRY
XCH A, @RO   ; PASS BACK TRANSLATED VALUE AND RESTORE
            ; ACC
RET          ; RETURN TO BACKGROUND PROGRAM
STPTBL DB 01101111B ; POSITION 0
DB 01011111B ; POSITION 1
DB 10011111B ; POSITION 2
DB 10101111B ; POSITION 3

```

The background program may reach this subroutine with several different calling sequences, all of which PUSH a value before calling the routine and POP the result after. A motor on Port 1 may be initialized by placing the desired position (zero) on the stack before calling the subroutine and outputting the results directly to a port afterwards.

Example 20—Sending and Receiving Data Parameters Via the Stack

```

CLR A
PUSH ACC
CALL NXTPOS
POP P1

```

If the position of the motor is determined by the contents of variable POSM1 (a byte in internal RAM) and the position of a second motor on Port 2 is determined by the data input to the low-order nibble of Port 2, a six-instruction sequence could update them both.

Example 21—Loading and Unloading Stack Direct from I/O Ports

```

POSM1 EQU 51
PUSH POSM1
CALL NXTPOS
POP P1
PUSH P2
CALL NXTPOS
POP P2

```

Data Pointer and Table Look-up Instructions — MOV, INC, MOVC, JMP

The data pointer can be loaded with a 16-bit value using the instruction `MOV DPTR, #data16`. The data used is stored in the second and third instruction bytes, high-order byte first. The data pointer is incremented by `INC DPTR`. A 16-bit increment is performed; an overflow from the low byte will carry into the high-order byte. Neither instruction affects any flags.

The `MOVC` (Move Constant) instructions (`MOVC A, @A+DPTR` and `MOVC A, @A+PC`) read into the accumulator bytes of data from the program memory logical address space. Both use a form of indexed addressing: the former adds the unsigned eight-bit accumulator contents with the sixteen-bit data pointer register, and uses the resulting sum as the address from which the byte is fetched. A sixteen-bit addition is performed; a carry-out from the low-order eight bits may propagate through higher-order bits, but the contents of the `DPTR` are not altered. The latter form uses the incremented program counter as the "base" value instead of the `DPTR` (figure 17). Again, neither version affects the flags.

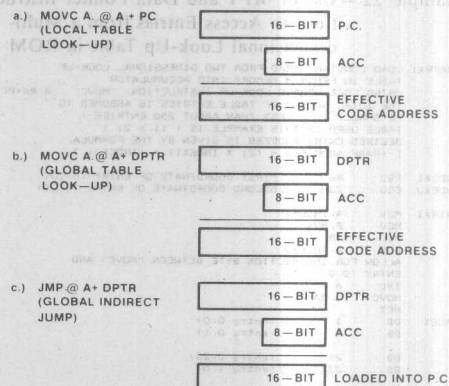


Figure 17. Operation of `MOVC` instructions

Each can be part of a three step sequence to access look-up tables in ROM. To use the DPTR-relative version, load the Data Pointer with the starting address of a look-up table; load the accumulator with (or compute) the index of the entry desired; and execute `MOVC A,@A+DPTR`. Unlike the similar `MOVP3` instructions in the 8048, the table may be located anywhere in program memory. The data pointer may be loaded with a constant for short tables. Or to allow more complicated data structures, or tables with more than 256 entries, the values for DPH and DPL may be computed or modified with the standard arithmetic instruction set.

The PC-relative version has the advantage of not affecting the data pointer. Again, a look-up sequence takes three steps: load the accumulator with the index; compensate for the offset from the look-up instruction to the start of the table by adding the number of bytes separating them to the accumulator; then execute the `MOVC A,@A+PC` instruction.

Let's look at a non-trivial situation where this instruction would be used. Some applications store large multi-dimensional look-up tables of dot matrix patterns, non-linear calibration parameters, and so on in a linear (one-dimensional) vector in program memory. To retrieve data from the tables, variables representing matrix indices must be converted to the desired entry's memory address. For a matrix of dimensions (MDIMEN x NDIMEN) starting at address BASE and respective indices INDEXI and INDEXJ, the address of element (INDEXI, INDEXJ) is determined by the formula,

$$\text{Entry Address} = \text{BASE} + (\text{NDIMEN} \times \text{INDEXI}) + \text{INDEXJ}$$

The code shown below can access any array with less than 255 entries (i.e., an 11x21 array with 231 elements). The table entries are defined using the Data Byte ("DB") directive, and will be contained in the assembly object code as part of the accessing subroutine itself.

Example 22—Use of MPY and Data Pointer Instructions to Access Entries from a Multi-dimensional Look-Up Table in ROM

```

;MATRIX1 LOAD CONSTANT READ FROM TWO DIMENSIONAL LOOK-UP
;TABLE IN PROGRAM MEMORY INTO ACCUMULATOR
;USING LOCAL TABLE LOOK-UP INSTRUCTION, "MOVC A,@A+PC"
;THE TOTAL NUMBER OF TABLE ENTRIES IS ASSUMED TO
;BE SMALL, I.E. LESS THAN ABOUT 250 ENTRIES
;TABLE USED IN THIS EXAMPLE IS ( 11 X 21 )
;DESIRED ENTRY ADDRESS IS GIVEN BY THE FORMULA,
;( BASE ADDRESS ) + ( 21 X INDEXI ) + ( INDEXJ ),
INDEXI EQU R6 ;FIRST COORDINATE OF ENTRY (0-10)
INDEXJ EQU R7H ;SECOND COORDINATE OF ENTRY (0-20)
MATRIX1 MOV A,INDEXI
MOV B,#21
MUL AB
ADD A,A,INDEXJ
;ALLOW FOR INSTRUCTION BYTE BETWEEN "MOVC" AND
;ENTRY (0,0)
INC A
MOVC A,@A+PC-01
RET
;BASE1
DB 1 ;entry 0,0)
DB 2 ;entry 0,1)
DB 21 ;entry 0,20)
DB 22 ;entry 1,0)
DB 42 ;entry 1,20)
DB 231 ;entry 10,20)

```

Figure 17. OpenMPY Instructions

There are several different means for branching to sections of code determined or selected at run time. (The single destination addresses incorporated into conditional and unconditional jumps are, of course, determined at assembly time). Each has advantages for different applications.

The most common is an N-way conditional jump based on some variable, with all of the potential destinations known at assembly time. One of a number of small routines is selected according to the value of an index variable determined while the program is running. The most efficient way to solve this problem is with the `MOVC` and an indirect jump instruction, using a short table of one byte offset values in ROM to indicate the relative starting addresses of the several routines.

`JMP @A+DPTR` is an instruction which performs an indirect jump to an address determined during program execution. The instruction adds the eight-bit unsigned accumulator contents with the contents of the sixteen-bit data pointer, just like `MOVC A,@A+DPTR`. The resulting sum is loaded into the program counter and is used as the address for subsequent instruction fetches. Again, a sixteen-bit addition is performed; a carry out from the low-order eight bits may propagate through the higher-order bits. In this case, neither the accumulator contents nor the data pointer is altered.

The example subroutine below reads a byte of RAM into the accumulator from one of four alternate address spaces, as selected by the contents of the variable `MEMSEL`. The address of the byte to be read is determined by the contents of `R0` (and optionally `R1`). It might find use in a printing terminal application, where four different model printers all use the same ROM code but use different types and sizes of buffer memory for different speeds and options.

Example 23—N-Way Branch and Computed Jump Instructions via `JMP @ADPTR`

```

MEMSEL EQU R3 ;SELECT RU-MEMO BASE, 0-3
;JUMP 01 OVER NEXT 128760 BYTES WILL BE
;JUMP 02 OVER NEXT 128760 BYTES WILL BE
;JUMP 03 OVER NEXT 128760 BYTES WILL BE
;JUMP 04 OVER NEXT 128760 BYTES WILL BE
JUMP_4 MOV A,MEMSEL
MOV DPTR,#JMP1BL
MOVC A,@A+DPTR
JMP A+DPTR
JMP1BL DB MEMSP0-JMP1BL
DB MEMSP1-JMP1BL
DB MEMSP2-JMP1BL
DB MEMSP3-JMP1BL
MEMSP0 MOV A,R0 ;READ FROM INTERNAL RAM
RET
MEMSP1 MOVX A,R0 ;READ FROM 256 BYTES OF EXTERNAL RAM
RET
MEMSP2 MOV DPL,R0
MOVX DPH,R1
MOVX A,@DPTR ;READ FROM 64K BYTES OF EXTERNAL RAM
RET
MEMSP3 MOV A,R1
ANL A,#07H
ANL P1,#11111000B
ORL P1,A
MOVX A,@P1 ;READ FROM 4K BYTES OF EXTERNAL RAM
RET

```

Note that this approach is suitable whenever the size of jump table plus the length of the alternate routines is less than 256 bytes. The jump table and routines may be located anywhere in program memory, independent of 256-byte program memory pages.

a.) RAM Bit Addresses.

RAM BYTE (MSB)	(LSB)							
7FH	7F	7E	7D	7C	7B	7A	79	78
2FH	77	76	75	74	73	72	71	70
2EH	6F	6E	6D	6C	6B	6A	69	68
2CH	67	66	65	64	63	62	61	60
2BH	5F	5E	5D	5C	5B	5A	59	58
2AH	57	56	55	54	53	52	51	50
29H	4F	4E	4D	4C	4B	4A	49	48
28H	47	46	45	44	43	42	41	40
27H	3F	3E	3D	3C	3B	3A	39	38
26H	37	36	35	34	33	32	31	30
25H	2F	2E	2D	2C	2B	2A	29	28
24H	27	26	25	24	23	22	21	20
23H	1F	1E	1D	1C	1B	1A	19	18
22H	17	16	15	14	13	12	11	10
21H	0F	0E	0D	0C	0B	0A	09	08
20H	07	06	05	04	03	02	01	00
1FH	Bank 3							
18H	Bank 2							
17H								
10H								
0FH	Bank 1							
08H	Bank 0							
07H								
00H								

b.) Hardware Register Bit Addresses.

Direct Byte Address	Bit Addresses (MSB)								(LSB)	Hardware Register Symbol
0FFH										
0F0H	F7	F6	F5	F4	F3	F2	F1	F0		B
0E0H	E7	E6	E5	E4	E3	E2	E1	E0		ACC
0D0H	D7	D6	D5	D4	D3	D2	D1	D0		PSW
0B8H	—	—	—	BC	BB	BA	B9	B8		IP
0B0H	B7	B6	B5	B4	B3	B2	B1	B0		P3
0A8H	AF	—	—	AC	AB	AA	A9	A8		IE
0A0H	A7	A6	A5	A4	A3	A2	A1	A0		P2
98H	9F	9E	9D	9C	9B	9A	99	98		SCON
90H	97	96	95	94	93	92	91	90		P1
88H	8F	8E	8D	8C	8B	8A	89	88		TCON
80H	87	86	85	84	83	82	81	80		P0

Figure 18. Bit Address Maps

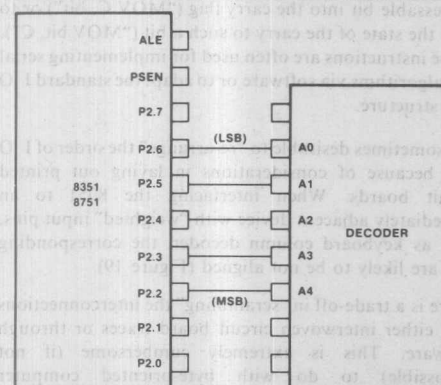


Figure 19. "Mismatch" Between I/O port and Decoder

Example 26—Re-ordering I/O Port Configuration

```

OUT_PZ  RRC  A  .MOVE ORIGINAL ACC 0 INTO CY
        MOV  P2 6.C .STORE CARRY TO PIN P26
        RRC  A  .MOVE ORIGINAL ACC 1 INTO CY
        MOV  P2 5.C .STORE CARRY TO PIN P25
        RRC  A  .MOVE ORIGINAL ACC 2 INTO CY
        MOV  P2 4.C .STORE CARRY TO PIN P24
        RRC  A  .MOVE ORIGINAL ACC 3 INTO CY
        MOV  P2 3.C .STORE CARRY TO PIN P23
        RRC  A  .MOVE ORIGINAL ACC 4 INTO CY
        MOV  P2 2.C .STORE CARRY TO PIN P22
        RET

```

Solving Combinatorial Logic Equations — ANL, ORL

Virtually all hardware designers are familiar with the problem of solving complex functions using combinatorial logic. The technologies involved may vary greatly, from multiple contact relay logic, vacuum tubes, TTL, or CMOS to more esoteric approaches like fluidics, but in each case the goal is the same: a Boolean (true false) function is computed on a number of Boolean variables.

AFN-01502A-28

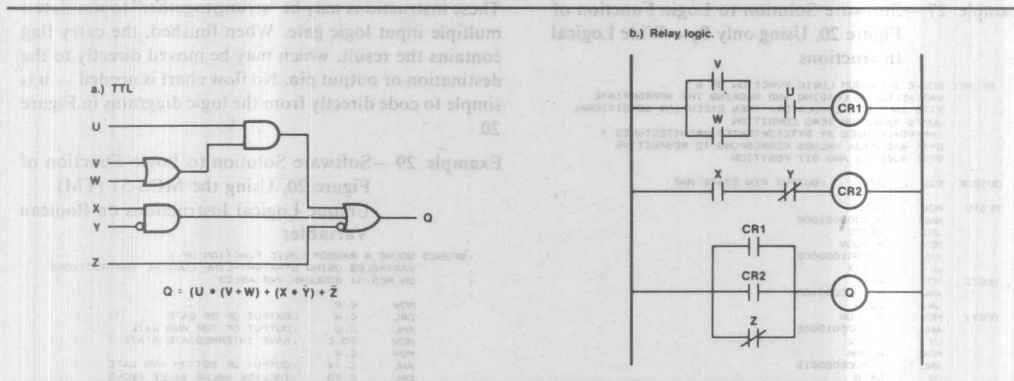


Figure 20. Implementations of Boolean functions

Figure 20 shows the logic diagram for an arbitrary function of six variables named U through Z using standard logic and relay logic symbols. Each is a solution of the equation.

$$Q = (U * (V + W)) + (X * Y) + Z$$

(While this equation could be reduced using Karnaugh Maps or algebraic techniques, that is not the purpose of this example. Even a minor change to the function equation would require re-reducing from scratch.)

Most digital computers can solve equations of this type with standard word-wide logical instructions and conditional jumps. Still, such software solutions seem somewhat sloppy because of the many paths through the program the computation can take.

Assume U and V are input pins being read by different input ports. W and X are status bits for two peripheral controllers (read as I/O ports), and Y and Z are software flags set or cleared earlier in the program. The end result must be written to an output pin on some third port.

For the sake of comparison we will implement this function with software drawn from three proper subsets of the MCS-51™ instruction set. The first two implementations follow the flow chart shown in Figure 21. Program flow would embark on a route down a test-and-branch tree and leaves either the "True" or "Not True" exit ASAP. These exits then write the output port with the data previously written to the same port with the result bit respectively one or zero.

In the first case, we assume there are no instructions for addressing individual bits other than special flags like the carry. This is typical of many older microprocessors and mainframe computers designed for number-crunching. MCS-51™ mnemonics are used here, though for most other machines the issue would be even further clouded by their use of operation-specific mnemonics like

INPUT, OUTPUT, LOAD, STORE, etc., instead of the universal MOV.

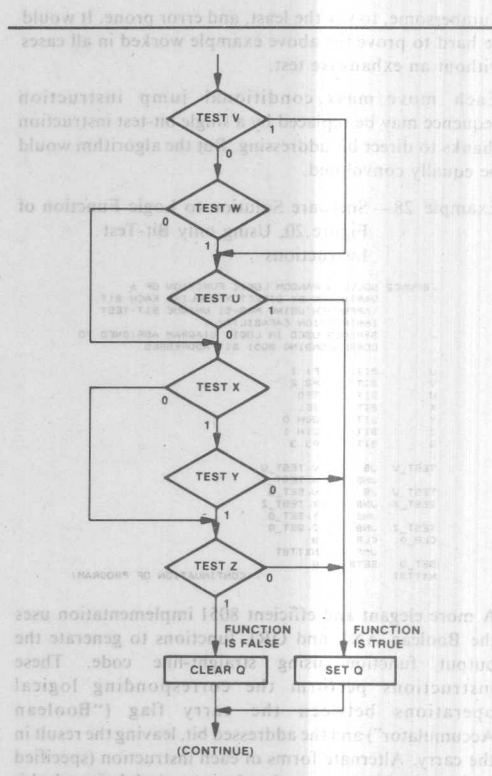


Figure 21. Flow chart for tree-branching logic implementation

Example 27—Software Solution to Logic Function of Figure 20, Using only Byte-Wide Logical Instructions

```

.BFUNC1 SOLVE A RANDOM LOGIC FUNCTION OF 6
; VARIABLES BY LOADING AND MASKING THE APPROPRIATE
; BITS IN THE ACCUMULATOR, THEN EXECUTING CONDITIONAL
; JUMPS BASED ON ZERO CONDITION
; (APPROACH USED BY BYTE-ORIENTED ARCHITECTURES)
; BYTE AND MASK VALUES CORRESPOND TO RESPECTIVE
; BYTE ADDRESS AND BIT POSITION
;
OUTBUF EQU 22H ; OUTPUT PIN STATE MAP
;
TESTV MOV A,P2
ANL A,#00000100B
JNZ TESTU
MOV A,TCON
ANL A,#00100000B
JZ TESTX
MOV A,P1
ANL A,#00000010B
JNZ SETQ
TESTX MOV A,TCON
ANL A,#00000100B
JZ TESTZ
MOV A,20H
ANL A,#00000001B
JZ SETQ
TESTZ MOV A,21H
ANL A,#00000010B
JZ SETQ
CLRQ MOV A,OUTBUF
ANL A,#11110111B
JMP OUTQ
SETQ MOV A,OUTBUF
ORL A,#00001000B
MOV OUTBUF,A
OUTQ MOV P3,A

```

Cumbersome, to say the least, and error prone. It would be hard to prove the above example worked in all cases without an exhaustive test.

Each move/mask/conditional jump instruction sequence may be replaced by a single bit-test instruction thanks to direct bit addressing. But the algorithm would be equally convoluted.

Example 28—Software Solution to Logic Function of Figure 20, Using only Bit-Test Instructions

```

.BFUNC2 SOLVE A RANDOM LOGIC FUNCTION OF 6
; VARIABLES BY DIRECTLY POLLING EACH BIT
; (APPROACH USING MCS-51 UNIQUE BIT-TEST
; INSTRUCTION CAPABILITY)
; SYMBOLS USED IN LOGIC DIAGRAM ASSIGNED TO
; CORRESPONDING 8051 BIT ADDRESSES
;
U BIT P1.1
V BIT P2.2
W BIT TFO
X BIT IE1
Y BIT 20H.0
Z BIT 21H.1
G BIT P3.3
;
TEST_V JB V,TEST_U
JNB W,TEST_X
TEST_U JB U,SET_Q
JNB X,TEST_Z
TEST_X JNB X,TEST_Z
JNB Y,SET_Q
TEST_Z JNB Z,SET_Q
CLR_Q CLR Q
JMP NXTTST
SET_Q SETB Q
NXTTST

```

A more elegant and efficient 8051 implementation uses the Boolean ANL and ORL functions to generate the output function using straight-line code. These instructions perform the corresponding logical operations between the carry flag ("Boolean Accumulator") and the addressed bit, leaving the result in the carry. Alternate forms of each instruction (specified in the assembly language by placing a slash before the bit-name) use the complement of the bit's state as the input operand.

These instructions may be "strung together" to simulate a multiple input logic gate. When finished, the carry flag contains the result, which may be moved directly to the destination or output pin. No flow chart is needed — it is simple to code directly from the logic diagrams in Figure 20.

Example 29—Software Solution to Logic Function of Figure 20, Using the MCS-51 (TM) Unique Logical Instructions on Boolean Variables

```

.BFUNC3 SOLVE A RANDOM LOGIC FUNCTION OF 6
; VARIABLES USING STRAIGHT-LINE LOGICAL INSTRUCTIONS
; ON MCS-51 BOOLEAN VARIABLES
;
MOV C,V ; OUTPUT OF OR GATE
ORL C,W ; OUTPUT OF TOP AND GATE
ANL C,U ; OUTPUT OF TOP AND GATE
MOV FO,C ; SAVE INTERMEDIATE STATE
MOV C,X ; OUTPUT OF BOTTOM AND GATE
ANL C,Y ; INCLUDE VALUE SAVED ABOVE
ORL C,FO ; INCLUDE LAST INPUT VARIABLE
ORL C,Z ; INCLUDE LAST INPUT VARIABLE
MOV G,C ; OUTPUT COMPUTED RESULT

```

Simplicity itself. Fast, flexible, reliable, easy to design, and easy to debug.

The Boolean features are useful and unique enough to warrant a complete Application Note of their own. Additional uses and ideas are presented in Application Note AP-70, *Using the Intel® MCS-51® Boolean Processing Capabilities*, publication number 121519.

5. ON-CHIP PERIPHERAL FUNCTION OPERATION AND INTERFACING

I/O Ports

The I/O port versatility results from the "quasi-bidirectional" output structure depicted in Figure 22. (This is effectively the structure of ports 1, 2, and 3 for normal I/O operations. On port 0 resistor R2 is disabled except during multiplexed bus operations, providing

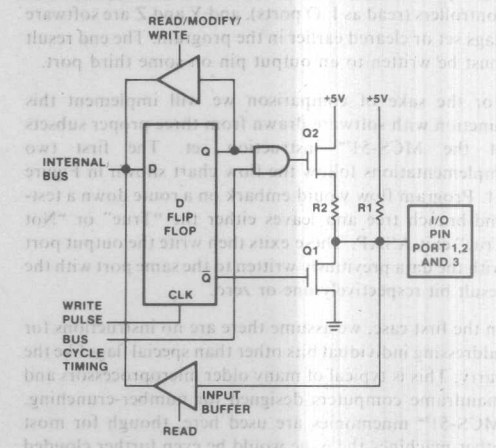


Figure 22. Pseudo-bidirectional I/O port circuitry

AFN-01502A-30

essentially open-collector outputs. For full electrical characteristics see the User's Manual.)

An output latch bit associated with each pin is updated by direct addressing instructions when that port is the destination. The latch state is buffered to the outside world by R1 and Q1, which may drive a standard TTL input. (In TTL terms, Q1 and R1 resemble an open collector output with a pull-up resistor to Vcc.)

R2 and Q2 represent an "active pull-up" device enabled momentarily when a 0 previously output changes to a 1. This "jerks" the output pin to a 1 level more quickly than the passive pull-up, improving rise-time significantly if the pin is driving a capacitive load. Note that the active pull-up is **only** activated on 0-to-1 transitions at the output latch (unlike the 8048, in which Q2 is activated whenever a 1 is written out).

Operations using an input port or pin as the source operand use the logic level of the pin itself, rather than the output latch contents. This level is affected by both the microcomputer itself and whatever device the pin is connected to externally. The value read is essentially the "OR-tied" function of Q1 and the external device. If the external device is high-impedence, such as a logic gate input or a three state output in the third state, then reading a pin will reflect the logic level previously output. To use a pin for input, the corresponding output latch must be set. The external device may then drive the pin with either a high or low logic signal. Thus the same port may be used as both input and output by writing ones to all pins used as inputs on output operations, and ignoring all pins used as output on an input operation.

In one operand instructions (INC, DEC, DJNZ and the Boolean CPL) the output latch rather than the input pin level is used as the source data. Similarly, two operand instructions using the port as both one source and the destination (ANL, ORL, XRL) use the output latches. This ensures that latch bits corresponding to pins used as inputs will not be cleared in the process of executing these instructions.

The Boolean operation JBC tests the output latch bit, rather than the input pin, in deciding whether or not to jump. Like the byte-wise logical operations, Boolean operations which modify individual pins of a port leave the other bits of the output latch unchanged.

A good example of how these modes may play together may be taken from the host-processor interface expected by an 8243 I/O expander. Even though the 8051 does not include 8048-type instructions for interfacing with an 8243, the parts can be interconnected (Figure 23) and the protocol may be emulated with simple software.

Example 30 — Mixing Parallel Output, Input, and Control Strokes on Port 2

```
IN8243  INP#1 DATA FROM AN 8243 I/O EXPANDER
        CONNECTED TO P23-P20
        P25 = P24 MIMIC CS = A PROG
        P27-P26 USED AS INPUTS
        PORT TO BE READ IN ACC

IN8243  OR  A #11010000D
        MOV  P0,A      OUTPUT INSTRUCTION CODE
        CLR  P1,4      FALLING EDGE OF PROG
        CPL  P2,00000111B  SET FOR INPUT
        MOV  A,P2      READ INP#1 DATA
        SET  P3,4      RETURN PROG HI-W
        SET  P0P2,0  DESELECT CHIP
```

Serial Port and Timer applications

Configuring the 8051's Serial Port for a given data rate and protocol requires essentially three short sections of software. On power-up or hardware reset the serial port and timer control words must be initialized to the appropriate values. Additional software is also needed in the transmit routine to load the serial port data register and in the receive routine to unload the data as it arrives.

This is best illustrated through an arbitrary example. Assume the 8051 will communicate with a CRT operating at 2400 baud (bits per second). Each character is transmitted as seven data bits, odd parity, and one stop bit. This results in a character rate of 2400/10=240 characters per second.

For the sake of clarity, the transmit and receive subroutines are driven by simple-minded software status polling code rather than interrupts. (It might help to refer back to Figures 7-9 showing the control word formats.) The serial port must be initialized to 8-bit UART mode (M0, M1=01), enabled to receive all messages (M2=0, REN=1). The flag indicating that the transmit register is free for more data will be artificially set in order to let the output software know the output register is available. This can all be set up with one instruction.

Example 31 — Serial Port Mode and Control Bits

```
SPINIT: MOV  SP,SPINIT
        MOV  SCON,#01010010B
        SETB TRSMOD
        SETB REN
```

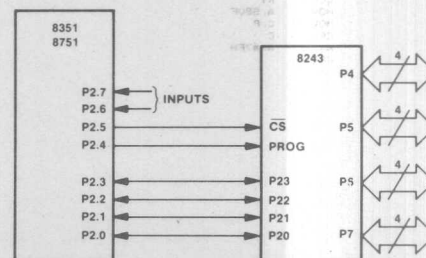


Figure 23. Connecting an 8051 with an 8243 I/O Expander

Timer 1 will be used in auto-reload mode as a data rate generator. To achieve a data rate of 2400 baud, the timer must divide the 1 MHz internal clock by $32 \times$ (desired data rate):

$$\frac{1 \times 10^6}{(32)(2400)}$$

which equals 13.02 rounded down to 13 instruction cycles. The timer must reload the value -13, or 0F3H. (ASM51 will accept both the signed decimal or hexadecimal representations.)

Example 32 — Initializing Timer Mode and Control Bits

```

;TINIT INITIALIZE TIMER 1 FOR
;      AUTO-RELOAD AT 32*2400 HZ
;      (TO USED AS GATED 16-BIT COUNTER )

TINIT  MOV     TCON,#11010010B
        MOV     TH1,#-13
        SETB    TR1

```

A simple subroutine to transmit the character passed to it in the accumulator must first compute the parity bit, insert it into the data byte, wait until the transmitter is available, output the character, and return. This is nearly as easy said as done.

Example 33—Code for UART Output, Adding Parity, Transmitter Loading

```

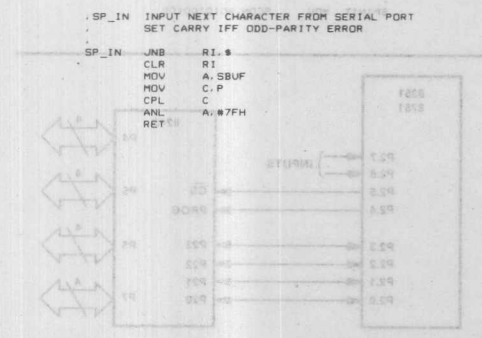
        SP_OUT    ADD ODD PARITY TO ACC AND
                    TRANSMIT WHEN SERIAL PORT READY.

        SP_OUT    MOV     C, P
                    CPL     ACC
                    MOV     ACC, 7 C
                    JNB     TI, $
                    CLR     R1
                    MOV     R1, SP_OUT.A

```

A simple minded routine to wait until a character is received, set the carry flag if there is an odd-parity error, and return the masked seven-bit code in the accumulator is equally short.

Example 34—Code for UART Reception and Parity Verification



6. SUMMARY

This Application Note has described the architecture, instruction set, and on-chip peripheral features of the first three members of the MCS-51™ microcomputer family. The examples used throughout were admittedly (and necessarily) very simple. Additional examples and techniques may be found in the MCS-51™ User's Manual and other application notes written for the MCS-48™ and MCS-51™ families.

Since its introduction in 1977, the MCS-48™ family has become the industry standard single-chip microcomputer. The MCS-51™ architecture expands the addressing capabilities and instruction set of its predecessor while ensuring flexibility for the future, and maintaining basic software compatibility with the past.

Designers already familiar with the 8048 or 8049 will be able to take with them the education and experience gained from past designs as ever-increasing system performance demands force them to move on to state-of-the-art products. Newcomers will find the power and regularity of the 8051 instruction set an advantage in streamlining both the learning and design processes.

Microcomputer system designers will appreciate the 8051 as basically a single-chip solution to many problems which previously required board-level computers. Designers of real-time control systems will find the high execution speed, on-chip peripherals, and interrupt capabilities vital in meeting the timing constraints of products previously requiring discrete logic designs. And designers of industrial controllers will be able to convert ladder diagrams directly from tested-and-true TTL or relay-logic designs to microcomputer software, thanks to the unique Boolean processing capabilities.

It has not been the intent of this note to gloss over the difficulty of designing microcomputer-based systems. To be sure, the hardware and software design aspects of any new computer system are nontrivial tasks. However, the system speed and level of integration of the MCS-51™ microcomputers, the power and flexibility of the instruction set, and the sophisticated assembler and other support products combine to give both the hardware and software designer as much of a head start on the problem as possible.

blank motor is either turned on or off, and so forth. For each control situation the most significant input variables are selected and the corresponding outputs of the MCS-31 architecture in its complete hardware support for one-bit or Boolean variables (named in support of Multigranular (Coarse) Models) as a separate motor or Multigranular (Coarse) Models.

Using the Intel M

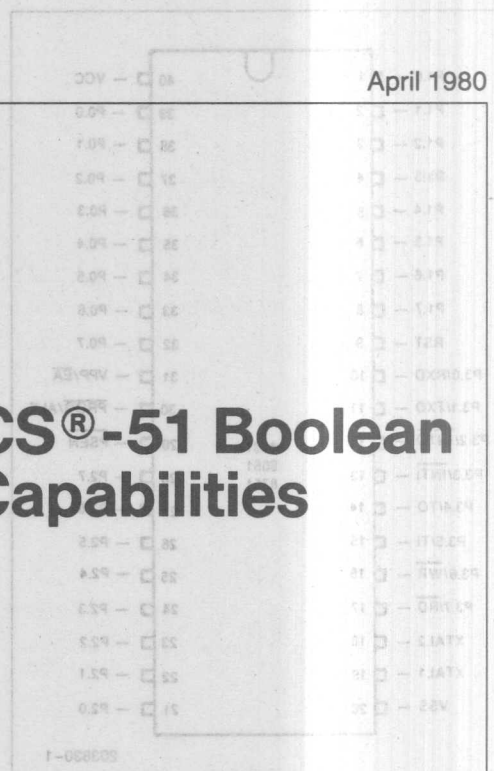
ERRON	Program Memory	Program Memory	External Program Memory	Program Memory (Int.)	Data Memory	Inst. Cycle Time	Input Output Pins	Interrupt Sources	Red. Banks
6748	8048	8048	8048	2K 4K	128	1.36 μ s	27	2	2
—	8049	8049	8049	2K 4K	128	1.0 μ s	32	2	2
6761	8061	8061	8061	4K 64K	128	1.0 μ s	32	2	4

Using the Intel MCS[®]-51 Boolean Processing Capabilities

JOHN WHARTON

MICROCONTROLLER APPLICATIONS

April 1980



1.0 INTRODUCTION

The Intel microcontroller family now has three new members: the Intel® 8031, 8051, and 8751 single-chip microcomputers. These devices, shown in Figure 1, will allow whole new classes of products to benefit from recent advances in Integrated Electronics. Thanks to Intel's new HMOS technology, they provide larger program and data memory spaces, more flexible I/O and peripheral capabilities, greater speed, and lower system cost than any previous-generation single-chip microcomputer.

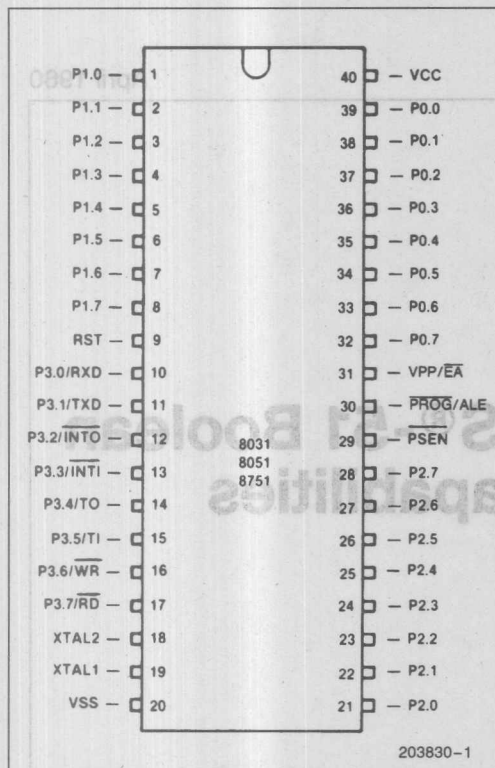


Figure 1. 8051 Family Pinout Diagram

Table 1. Features of Intel's Single-Chip Microcomputers

EPROM Program Memory	ROM Program Memory	External Program Memory	Program Memory (Int/Max)	Data Memory (Bytes)	Instr. Cycle Time	Input/ Output Pins	Interrupt Sources	Reg. Banks
8748	8048	8035	1K 4K	64	2.5 μ s	27	2	2
—	8049	8039	2K 4K	128	1.36 μ s	27	2	2
8751	8051	8031	4K 64K	128	1.0 μ s	32	5	4

Table 1 summarizes the quantitative differences between the members of the MCS®-48 and 8051 families. The 8751 contains 4K bytes of EPROM program memory fabricated on-chip, while the 8051 replaces the EPROM with 4K bytes of lower-cost mask-programmed ROM. The 8031 has no program memory on-chip; instead, it accesses up to 64K bytes of program memory from external memory. Otherwise, the three new family members are identical. Throughout this Note, the term "8051" will represent all members of the 8051 Family, unless specifically stated otherwise.

The CPU in each microcomputer is one of the industry's fastest and most efficient for numerical calculations on byte operands. But controllers often deal with bits, not bytes: in the real world, switch contacts can only be open or closed, indicators should be either lit or dark, motors are either turned on or off, and so forth. For such control situations the most significant aspect of the MCS®-51 architecture is its complete hardware support for one-bit, or *Boolean* variables (named in honor of Mathematician George Boole) as a separate data type.

The 8051 incorporates a number of special features which support the direct manipulation and testing of individual bits and allow the use of single-bit variables in performing logical operations. Taken together, these features are referred to as the MCS-51 *Boolean Processor*. While the bit-processing capabilities alone would be adequate to solve many control applications, their true power comes when they are used in conjunction with the microcomputer's byte-processing and numerical capabilities.

Many concepts embodied by the Boolean Processor will certainly be new even to experienced microcomputer system designers. The purpose of this Application Note is to explain these concepts and show how they are used.

For detailed information on these parts refer to the **Intel Microcontroller Handbook**, order number 210918. The instruction set, assembly language, and use of the 8051 assembler (ASM51) are further described in the **MCS®-51 Macro Assembler User's Guide for DOS Systems**, order number 122753.

2.0 BOOLEAN PROCESSOR OPERATION

The Boolean Processing capabilities of the 8051 are based on concepts which have been around for some time. Digital computer systems of widely varying designs all have four functional elements in common (Figure 2):

- a central processor (CPU) with the control, timing, and logic circuits needed to execute stored instructions;
- a memory to store the sequence of instructions making up a program or algorithm;
- data memory to store variables used by the program;
- and
- some means of communicating with the outside world.

The CPU usually includes one or more accumulators or special registers for computing or storing values during program execution. The instruction set of such a processor generally includes, at a minimum, operation classes to perform arithmetic or logical functions on program variables, move variables from one place to another, cause program execution to jump or conditionally branch based on register or variable states, and instructions to call and return from subroutines. The program and data memory functions sometimes share a single memory space, but this is not always the case. When the address spaces are separated, program and data memory need not even have the same basic word width.

A digital computer's flexibility comes in part from combining simple fast operations to produce more com-

plex (albeit slower) ones, which in turn link together eventually solving the problem at hand. A four-bit CPU executing multiple precision subroutines can, for example, perform 64-bit addition and subtraction. The subroutines could in turn be building blocks for floating-point multiplication and division routines. Eventually, the four-bit CPU can simulate a far more complex "virtual" machine.

In fact, *any* digital computer with the above four functional elements can (given time) complete *any* algorithm (though the proverbial room full of chimpanzees at word processors might first re-create Shakespeare's classics and this Application Note)! This fact offers little consolation to product designers who want programs to run as quickly as possible. By definition, a real-time control algorithm *must* proceed quickly enough to meet the preordained speed constraints of other equipment.

One of the factors determining how long it will take a microcomputer to complete a given chore is the number of instructions it must execute. What makes a given computer architecture particularly well- or poorly-suited for a class of problems is how well its instruction set matches the tasks to be performed. The better the "primitive" operations correspond to the steps taken by the control algorithm, the lower the number of instructions needed, and the quicker the program will run. All else being equal, a CPU supporting 64-bit arithmetic directly could clearly perform floating-point math faster than a machine bogged-down by multiple-precision subroutines. In the same way, direct support for bit manipulation naturally leads to more efficient programs handling the binary input and output conditions inherent in digital control problems.

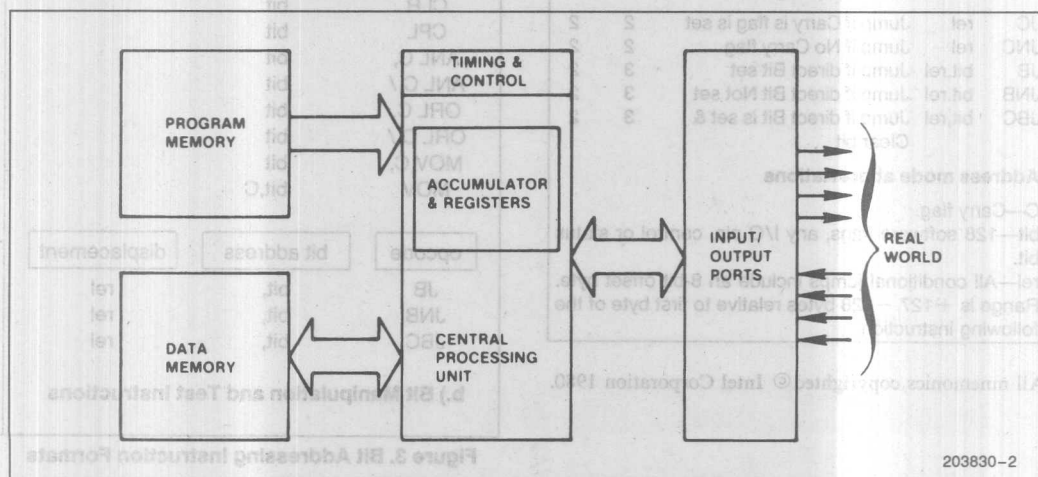


Figure 2. Block Diagram for Abstract Digital Computer

Processing Elements

The introduction stated that the 8051's bit-handling capabilities alone would be sufficient to solve some control applications. Let's see how the four basic elements of a digital computer—a CPU with associated registers, program memory, addressable data RAM, and I/O capability—relate to Boolean variables.

CPU. The 8051 CPU incorporates special logic devoted to executing several bit-wide operations. All told, there are 17 such instructions, all listed in Table 2. Not shown are 94 other (mostly byte-oriented) 8051 instructions.

Program Memory. Bit-processing instructions are fetched from the same program memory as other arithmetic and logical operations. In addition to the instruc-

tions of Table 2, several sophisticated program control features like multiple addressing modes, subroutine nesting, and a two-level interrupt structure are useful in structuring Boolean Processor-based programs.

Boolean instructions are one, two, or three bytes long, depending on what function they perform. Those involving only the carry flag have either a single-byte opcode or an opcode followed by a conditional-branch destination byte (Figure 3a). The more general instructions add a "direct address" byte after the opcode to specify the bit affected, yielding two or three byte encodings (Figure 3b). Though this format allows potentially 256 directly addressable bit locations, not all of them are implemented in the 8051 family.

Table 2. MCS-51™ Boolean Processing Instruction Subset

Mnemonic	Description	Byte	Cyc
SETB C	Set Carry flag	1	1
SETB bit	Set direct Bit	2	1
CLR C	Clear Carry flag	1	1
CLR bit	Clear direct bit	2	1
CPL C	Complement Carry flag	1	1
CPL bit	Complement direct bit	2	1
MOV C,bit	Move direct bit to Carry flag	2	1
MOV bit,C	Move Carry flag to direct bit	2	2
ANL C,bit	AND direct bit to Carry flag	2	2
ANL C,bit	AND complement of direct bit to Carry flag	2	2
ORL C,bit	OR direct bit to Carry flag	2	2
ORL C,bit	OR complement of direct bit to Carry flag	2	2
JC rel	Jump if Carry is flag is set	2	2
JNC rel	Jump if No Carry flag	2	2
JB bit,rel	Jump if direct Bit set	3	2
JNB bit,rel	Jump if direct Bit Not set	3	2
JBC bit,rel	Jump if direct Bit is set & Clear bit	3	2

Address mode abbreviations

C—Carry flag.

bit—128 software flags, any I/O pin, control or status bit.

rel—All conditional jumps include an 8-bit offset byte. Range is +127 – 128 bytes relative to first byte of the following instruction.

All mnemonics copyrighted © Intel Corporation 1980.

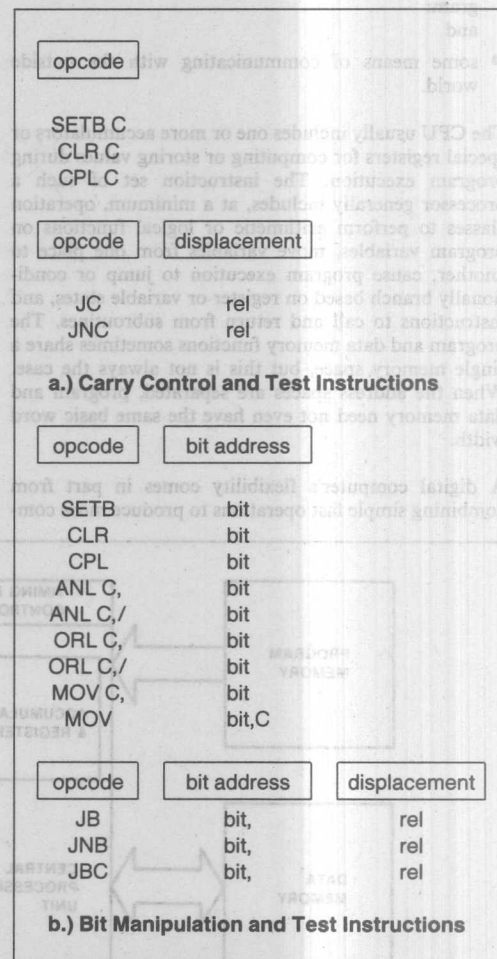


Figure 3. Bit Addressing Instruction Formats

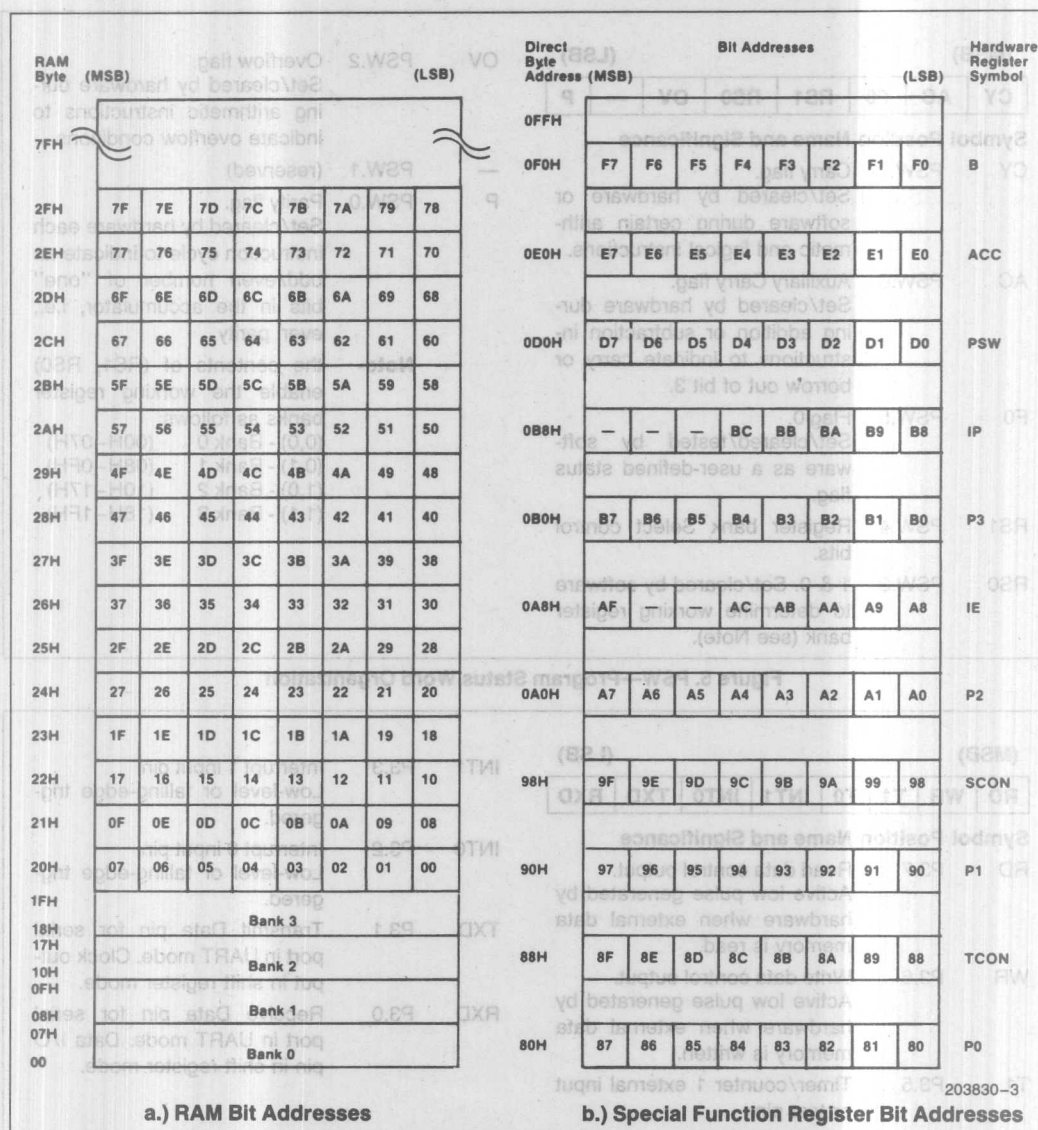


Figure 4. Bit Address Maps

Data Memory. The instructions in Figure 3b can operate directly upon 144 general purpose bits forming the Boolean processor "RAM." These bits can be used as software flags or to store program variables. Two operations and instructions use the CPU's carry flag ("C") as a special one-bit register: in a sense, the carry is a "Boolean accumulator" for logical operations and data transfers.

Input/Output. All 32 I/O pins can be addressed as individual inputs, outputs, or both, in any combination. Any pin can be a control strobe output, status (Test) input, or serial I/O link implemented via software. An additional 33 individually addressable bits reconfigure, control, and monitor the status of the CPU and all on-chip peripheral functions (timer counters, serial port modes, interrupt logic, and so forth).

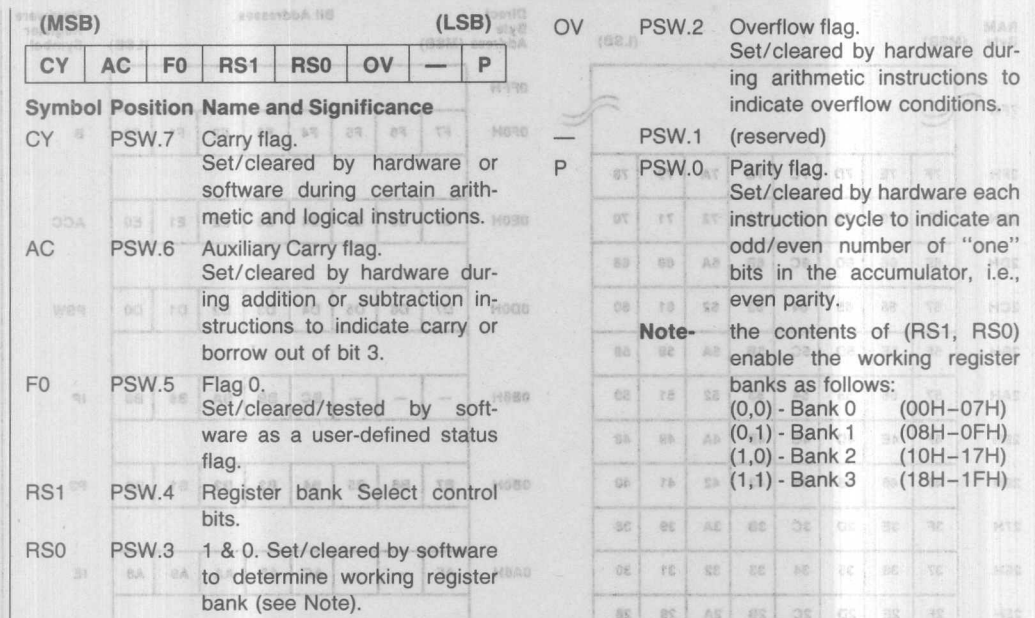


Figure 5. PSW—Program Status Word Organization

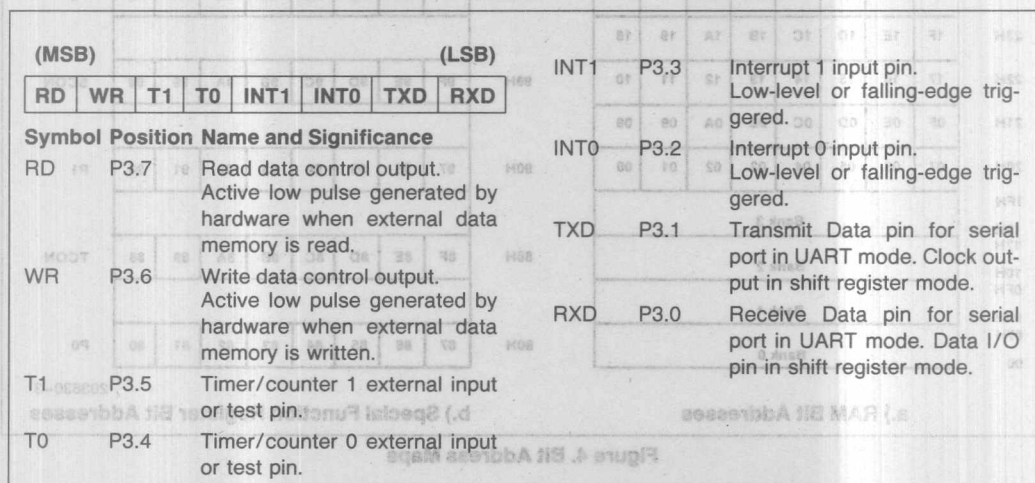


Figure 6. P3—Alternate I/O Functions of Port 3

Direct Bit Addressing

The most significant bit of the direct address byte selects one of two groups of bits. Values between 0 and 127 (00H and 7FH) define bits in a block of 32 bytes of on-chip RAM, between RAM addresses 20H and 2FH (Figure 4a). They are numbered consecutively from the lowest-order byte's lowest-order bit through the highest-order byte's highest-order bit.

Bit addresses between 128 and 255 (80H and 0FFH) correspond to bits in a number of special registers, mostly used for I/O or peripheral control. These positions are numbered with a different scheme than RAM: the five high-order address bits match those of the register's own address, while the three low-order bits identify the bit position within that register (Figure 4b).

Notice the column labeled "Symbol" in Figure 5. Bits with special meanings in the PSW and other registers have corresponding symbolic names. General-purpose (as opposed to carry-specific) instructions may access the carry like any other bit by using the mnemonic CY in place of C, P0, P1, P2, and P3 are the 8051's four I/O ports: secondary functions assigned to each of the eight pins of P3 are shown in Figure 6.

Figure 7 shows the last four bit addressable registers. TCON (Timer Control) and SCON (Serial port Control) control and monitor the corresponding peripherals, while IE (Interrupt Enable) and IP (Interrupt Priority) enable and prioritize the five hardware interrupt sources. Like the reserved hardware register addresses,

the five bits not implemented in IE and IP should not be accessed: they can *not* be used as software flags.

Addressable Register Set. There are 20 special function registers in the 8051, but the advantages of bit addressing only relate to the 11 described below. Five potentially bit-addressable register addresses (0C0H, 0C8H, 0D8H, 0E8H, & 0F8H) are being reserved for possible future expansion in microcomputers based on the MCS-51 architecture. Reading or writing non-existent registers in the 8051 series is pointless, and may cause unpredictable results. Byte-wide logical operations can be used to manipulate bits in all *non-bit* addressable registers and RAM.

Symbol Position Name and Significance		(MSB)				(LSB)			
SM0	SM1	SM2	SM3	REN	T88	T89	T90	T91	RI
SCON.0 Serial port Mode control bit 0. Self-cleared by software (see note).	SCON.1 Serial port Mode control bit 1. Self-cleared by software (see note).	SCON.2 Serial port Mode control bit 2. Set by software to disable reception of frames for which bit 3 is zero.	SCON.3 Receiver Enable control bit. Self-cleared by software to enable/disable serial data reception.	REN	SCON.4 Transmitt Bit 8. Self-cleared by hardware to determine state of ninth data bit transmitted in 9-bit UART mode.	SCON.5	SCON.6	SCON.7	SCON.8
Note: the state of (SM0, SM1) selects: (0,0)—Shift register I/O expansion. (0,1)—9-bit UART, variable data rate. (1,0)—9-bit UART, fixed data rate. (1,1)—9-bit UART, variable data rate.									
b.) SCON—Serial Port Control/Status Register									
a.) TCON—Timer/Counter Control/Status Register									

Figure 7. Peripheral Configuration Registers

(MSB)							(LSB)
TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0
Symbol Position Name and Significance							
TF1	TCON.7	Timer 1 overflow Flag. Set by hardware on timer/counter overflow. Cleared when interrupt processed.					
TR1	TCON.6	Timer 1 Run control bit. Set/cleared by software to turn timer/counter on/off.					
TF0	TCON.5	Timer 0 overflow Flag. Set by hardware on timer/counter overflow. Cleared when interrupt processed.					
TR0	TCON.4	Timer 0 Run control bit. Set/cleared by software to turn timer/counter on/off.					

a.) TCON—Timer/Counter Control/Status Register

(MSB)							(LSB)
SM0	SM1	SM2	REN	TB8	RB8	TI	RI
Symbol Position Name and Significance							
SM0	SCON.7	Serial port Mode control bit 0. Set/cleared by software (see note).					
SM1	SCON.6	Serial port Mode control bit 1. Set/cleared by software (see note).					
SM2	SCON.5	Serial port Mode control bit 2. Set by software to disable reception of frames for which bit 8 is zero.					
REN	SCON.4	Receiver Enable control bit. Set/cleared by software to enable/disable serial data reception.					
TB8	SCON.3	Transmit Bit 8. Set/cleared by hardware to determine state of ninth data bit transmitted in 9-bit UART mode.					

b.) SCON—Serial Port Control/Status Register

IE1	TCON.3	Interrupt 1 Edge flag. Set by hardware when external interrupt edge detected. Cleared when interrupt processed.					
IT1	TCON.2	Interrupt 1 Type control bit. Set/cleared by software to specify falling edge/low level triggered external interrupts.					
IE0	TCON.1	Interrupt 0 Edge flag. Set by hardware when external interrupt edge detected. Cleared when interrupt processed.					
IT0	TCON.0	Interrupt 0 Type control bit. Set/cleared by software to specify falling edge/low level triggered external interrupts.					

a.) SCON—Serial Port Control/Status Register

RB8	SCON.2	Receive Bit 8. Set/cleared by hardware to indicate state of ninth data bit received.					
TI	SCON.1	Transmit Interrupt flag. Set by hardware when byte transmitted. Cleared by software after servicing.					
RI	SCON.0	Receive Interrupt flag. Set by hardware when byte received. Cleared by software after servicing.					

Note-

the state of (SM0, SM1) selects:

(0,0)—Shift register I/O expansion.

(0,1)—8-bit UART, variable data rate.

(1,0)—9-bit UART, fixed data rate.

(1,1)—9-bit UART, variable data rate.

Figure 7. Peripheral Configuration Registers

(MSB)			(LSB)				
EA	—	—	ES	ET1	EX1	ET1	EX0

Symbol Position Name and Significance

EA	IE.7	Enable All control bit. Cleared by software to disable all interrupts, independent of the state of IE.4–IE.0.
—	IE.6	(reserved)
—	IE.5	(reserved)
ES	IE.4	Enable Serial port control bit. Set/cleared by software to enable/disable interrupts from TI or RI flags.
ET1	IE.3	Enable Timer 1 control bit. Set/cleared by software to enable/disable interrupts from timer/counter 1.

c.) IE—Interrupt Enable Register

(MSB)			(LSB)				
—	—	—	PS	PT1	PX1	PT0	PX0

Symbol Position Name and Significance

—	IP.7	(reserved)
—	IP.6	(reserved)
—	IP.5	(reserved)
PS	IP.4	Serial port Priority control bit. Set/cleared by software to specify high/low priority interrupts for Serial port.
PT1	IP.3	Timer 1 Priority control bit. Set/cleared by software to specify high/low priority interrupts for timer/counter 1.

d.) IP—Interrupt Priority Control Register

EX1	IE.2	Enable External interrupt 1 control bit. Set/cleared by software to enable/disable interrupts from INT1.
ET0	IE.1	Enable Timer 0 control bit. Set/cleared by software to enable/disable interrupts from timer/counter 0.
EX0	IE.0	Enable External interrupt 0 control bit. Set/cleared by software to enable/disable interrupts from INTO.

PX1	IP.2	External interrupt 1 Priority control bit. Set/cleared by software to specify high/low priority interrupts for INT1.
PT0	IP.1	Timer 0 Priority control bit. Set/cleared by software to specify high/low priority interrupts for timer/counter 0.
PX0	IP.0	External interrupt 0 Priority control bit. Set/cleared by software to specify high/low priority interrupts for INTO.

Figure 7. Peripheral Configuration Registers (Continued)

The accumulator and B registers (A and B) are normally involved in byte-wide arithmetic, but their individual bits can also be used as 16 general software flags. Added with the 128 flags in RAM, this gives 144 general purpose variables for bit-intensive programs. The program status word (PSW) in Figure 5 is a collection of flags and machine status bits including the carry flag itself. Byte operations acting on the PSW can therefore affect the carry.

Instruction Set

Having looked at the bit variables available to the Boolean Processor, we will now look at the four classes of

instructions that manipulate these bits. It may be helpful to refer back to Table 2 while reading this section.

State Control. Addressable bits or flags may be set, cleared, or logically complemented in one instruction cycle with the two-byte instructions SETB, CLR, and CPL. (The “B” affixed to SETB distinguishes it from the assembler “SET” directive used for symbol definition.) SETB and CLR are analogous to loading a bit with a constant: 1 or 0. Single byte versions perform the same three operations on the carry.

The MCS-51 assembly language specifies a bit address in any of three ways:

- by a number or expression corresponding to the direct bit address (0–255):

- by the name or address of the register containing the bit, the *dot operator* symbol (a period: "."), and the bit's position in the register (7-0):
- in the case of control and status registers, by the predefined assembler symbols listed in the first columns of Figures 5-7.

Bits may also be given user-defined names with the assembler "BIT" directive and any of the above techniques. For example, bit 5 of the PSW may be cleared by any of the four instructions.

```

USR_FLG BIT PSW.5 ; User Symbol Definition
....
CLR OD5H ; Absolute Addressing
CLR PSW.5 ; Use of Dot Operator
CLR FO ; Pre-Defined Assembler
; Symbol
CLR USR_FLG ; User-Defined Symbol

```

Data Transfers. The two-byte MOV instructions can transport any addressable bit to the carry in one cycle, or copy the carry to the bit in two cycles. A bit can be moved between two arbitrary locations via the carry by combining the two instructions. (If necessary, push and pop the PSW to preserve the previous contents of the carry.) These instructions can replace the multi-instruction sequence of Figure 8, a program structure appearing in controller applications whenever flags or outputs are conditionally switched on or off.

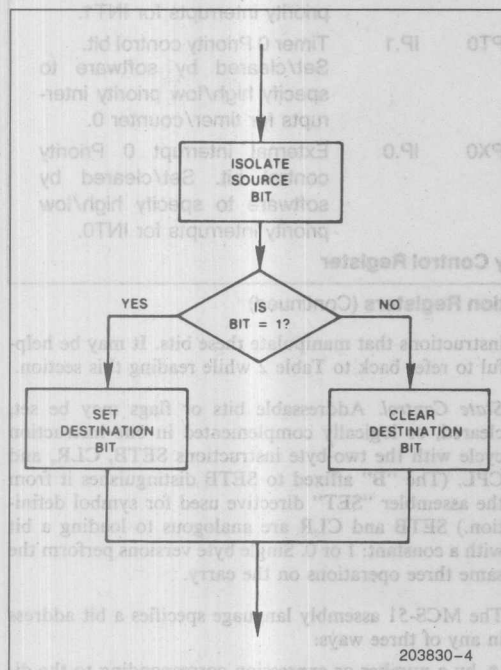


Figure 8. Bit Transfer Instruction Operation

Logical Operations. Four instructions perform the logical-AND and logical-OR operations between the carry and another bit, and leave the results in the carry. The instruction mnemonics are ANL and ORL; the absence or presence of a slash mark ("/") before the source operand indicates whether to use the positive-logic value or the logical complement of the addressed bit. (The source operand itself is never affected.)

Bit-test Instructions. The conditional jump instructions "JC rel" (Jump on Carry) and "JNC rel" (Jump on Not Carry) test the state of the carry flag, branching if it is a one or zero, respectively. (The letters "rel" denote relative code addressing.) The three-byte instructions "JB bit.rel" and "JNB bit.rel" (Jump on Bit and Jump on Not Bit) test the state of *any* addressable bit in a similar manner. A fifth instruction combines the Jump on Bit and Clear operations. "JBC bit.rel" conditionally branches to the indicated address, then clears the bit in the same two cycle instruction. This operation is the same as the MCS-48 "JTF" instructions.

All 8051 conditional jump instructions use program counter-relative addressing, and all execute in two cycles. The last instruction byte encodes a signed displacement ranging from -128 to +127. During execution, the CPU adds this value to the incremented program counter to produce the jump destination. Put another way, a conditional jump to the immediately following instruction would encode 00H in the offset byte.

A section of program or subroutine written using only relative jumps to nearby addresses will have the same machine code independent of the code's location. An assembled routine may be repositioned anywhere in memory, even crossing memory page boundaries, without having to modify the program or recompute destination addresses. To facilitate this flexibility, there is an unconditional "Short Jump" (SJMP) which uses relative addressing as well. Since a programmer would have quite a chore trying to compute relative offset values from one instruction to another, ASM51 automatically computes the displacement needed given only the destination address or label. An error message will alert the programmer if the destination is "out of range."

The so-called "Bit Test" instructions implemented on many other microprocessors simply perform the logical-AND operation between a byte variable and a constant mask, and set or clear a zero flag depending on the result. This is essentially equivalent to the 8051 "MOV C.bit" instruction. A second instruction is then needed to conditionally branch based on the state of the zero flag. This does *not* constitute abstract bit-addressing in the MCS-51 sense. A flag exists only as a field

within a register; to reference a bit the programmer must know and specify both the encompassing register and the bit's position therein. This constraint severely limits the flexibility of symbolic bit addressing and reduces the machine's code-efficiency and speed.

Interaction with Other Instructions. The carry flag is also affected by the instructions listed in Table 3. It can be rotated through the accumulator, and altered as a side effect of arithmetic instructions. Refer to the User's Manual for details on how these instructions operate.

Simple Instruction Combinations

By combining general purpose bit operations with certain addressable bits, one can "custom build" several hundred useful instructions. All eight bits of the PSW can be tested directly with conditional jump instructions to monitor (among other things) parity and overflow status. Programmers can take advantage of 128 software flags to keep track of operating modes, resource usage, and so forth.

The Boolean instructions are also the most efficient way to control or reconfigure peripheral and I/O registers. All 32 I/O lines become "test pins," for example, tested by conditional jump instructions. Any output pin can be toggled (complemented) in a single instruction cycle. Setting or clearing the Timer Run flags (TR0 and TR1) turn the timer/counters on or off; polling the same flags elsewhere lets the program determine if a timer is running. The respective overflow flags (TF0 and TF1) can be tested to determine when the desired period or count has elapsed, then cleared in preparation for the next repetition. (For the record, these bits are all part of the TCON register, Figure 7a. Thanks to symbolic bit addressing, the programmer only needs to remember the mnemonic associated with each function. In other words, don't bother memorizing control word layouts.)

In the MCS-48 family, instructions corresponding to some of the above functions require specific opcodes. Ten different opcodes serve to clear/complement the software flags F0 and F1, enable/disable each interrupt, and start/stop the timer. In the 8051 instruction set, just three opcodes (SETB, CLR, CPL) with a direct bit address appended perform the same functions. Two test instructions (JB and JNB) can be combined with bit addresses to test the software flags, the 8048 I/O pins T0, T1, and INT, and the eight accumulator bits, replacing 15 more different instructions.

Table 4a shows how 8051 programs implement software flag and machine control functions associated with special opcodes in the 8048. In every case the MCS-51 solution requires the same number of machine cycles, and executes 2.5 times faster.

Table 3. Other Instructions Affecting the Carry Flag

Mnemonic	Description	Byte	Cyc
ADD A,Rn	Add register to Accumulator	1	1
ADD A,direct	Add direct byte to Accumulator	2	1
ADD A,@Ri	Add indirect RAM to Accumulator	1	1
ADD A,#data	Add immediate data to Accumulator	2	1
ADDC A,Rn	Add register to Accumulator with Carry flag	1	1
ADDC A,direct	Add direct byte to Accumulator with Carry flag	2	1
ADDC A,@Ri	Add indirect RAM to Accumulator with Carry flag	1	1
ADDC A,#data	Add immediate data to Acc with Carry flag	2	1
SUBB A,Rn	Subtract register from Accumulator with borrow	1	1
SUBB A,direct	Subtract direct byte from Acc with borrow	2	1
SUBB A,@Ri	Subtract indirect RAM from Acc with borrow	1	1
SUBB A,#data	Subtract immediate data from Acc with borrow	2	1
MUL AB	Multiply A & B	1	4
DIV AB	Divide A by B	1	4
DA A	Decimal Adjust Accumulator	1	1
RLC A	Rotate Accumulator Left through the Carry flag	1	1
RRC A	Rotate Accumulator Right through Carry flag	1	1
CJNE A,direct,rel	Compare direct byte to Acc & Jump if Not Equal	3	2
CJNE A,#data,rel	Compare immediate to Acc & Jump if Not Equal	3	2
CJNE Rn,#data,rel	Compare immed to register & Jump if Not Equal	3	2
CJNE @Ri,#data,rel	Compare immed to indirect & Jump if Not Equal	3	2

All mnemonics copyrighted © Intel Corporation 1980.

Table 4a. Contrasting 8048 and 8051 Bit Control and Testing Instructions

8048					8x51				
Instruction		Bytes	Cycles	μSec	Instruction		Bytes	Cycles & μSec	
Flag Control									
CLR	C	1	1	2.5	CLR	C	1	1	
CPL	F0	1	1	2.5	CPL	F0	2	1	
Flag Testing									
JNC	offset	2	2	5.0	JNC	rel	2	2	
JF0	offset	2	2	5.0	JB	F0.rel	3	2	
JB7	offset	2	2	5.0	JB	ACC.7.rel	3	2	
Peripheral Polling									
JT0	offset	2	2	5.0	JB	T0.rel	3	2	
JN1	offset	2	2	5.0	JNB	INT0.rel	3	2	
JTF	offset	2	2	5.0	JBC	TF0.rel	3	2	
Machine and Peripheral Control									
STRT	T	1	1	2.5	SETB	TR0	2	1	
EN	1	1	1	2.5	SETB	EX0	2	1	
DIS	TCNT1	1	1	2.5	CLR	ET0	2	1	

Table 4b. Replacing 8048 Instruction Sequences with Single 8x51 Instructions

8048				8051							
Instruction		Bytes	Cycles	μSec	Instruction		Bytes	Cycles & μSec			
Flag Control											
Set carry											
CLR	C	= 2	2	5.0	SETB	C	1	1			
CPL	C										
Set Software Flag											
CLR	F0	= 2	2	5.0	SETB	F0	2	1			
CPL	F0										
Turn Off Output Pin											
ANL	P1.#0FBH	= 2	2	5.0	CLR	P1.2	2	1			
Complement Output Pin											
IN	A.P1	= 4	6	15.0							
XRL	A.#04H										
OUTL	P1.A							CPL	P1.2	2	1
Clear Flag in RAM											
MOV	R0.#FLGADR	= 6	6	15.0							
MOV	A.@R0							CLR	USER_FLG	2	1
ANL	A.#FLGMASK										
MOV	@R0.A										

Table 4b. Replacing 8048 Instruction Sequences with Single 8x51 Instructions (Continued)

8048 Instruction	Bytes	Cycles	μSec	8x51 Instruction	Bytes	Cycles & μSec
Flag Testing: Jump if Software Flag is 0 JF0 \$+4 JMP offset = 4	4	4	10.0	JNB F0.rel	3	2
Jump if Accumulator bit is 0 CPL A JB7 offset CPL A	4	4	10.0	JNB ACC.7.rel	3	2
Peripheral Polling Test if Input Pin is Grounded IN A.P1 CPL A JB3 offset = 4	4	5	12.5	JNB P1.3.rel	3	2
Test if Interrupt Pin is High JN1 \$+4 JMP offset = 4	4	4	10.0	JB INT0.rel	3	2

3.0 BOOLEAN PROCESSOR APPLICATIONS

So what? Then what does all this buy you?

Qualitatively, nothing. All the same capabilities could be (and often have been) implemented on other machines using awkward sequences of other basic operations. As mentioned earlier, any CPU can solve any problem given enough time.

Quantitatively, the differences between a solution allowed by the 8051 and those required by previous architectures are numerous. What the 8051 Family buys you is a faster, cleaner, lower-cost solution to micro-controller applications.

The opcode space freed by condensing many specific 8048 instructions into a few general operations has been used to add new functionality to the MCS-51 architecture—both for byte and bit operations. 144 software flags replace the 8048's two. These flags (and the carry) may be directly set, not just cleared and complemented, and all can be tested for either state, not just one. Operating mode bits previously inaccessible may be read, tested, or saved. Situations where the 8051 instruction set provides new capabilities are contrasted with 8048 instruction sequences in Table 4b. Here the 8051 speed advantage ranges from 5x to 15x!

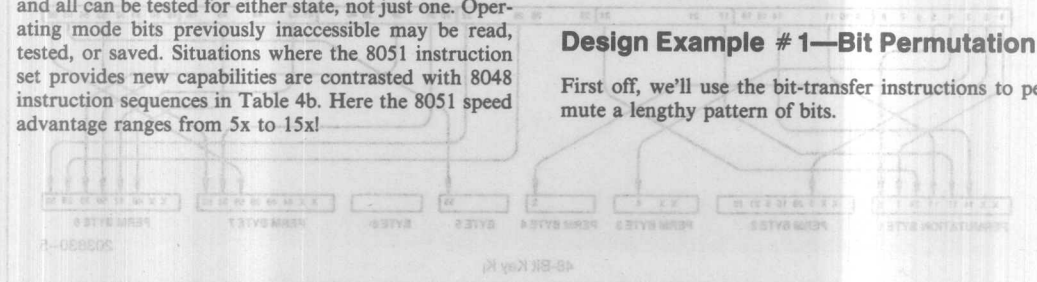
Combining Boolean and byte-wide instructions can produce great synergy. An MCS-51 based application will prove to be:

- simpler to write since the architecture correlates more closely with the problems being solved:
- easier to debug because more individual instructions have no unexpected or undesirable side-effects:
- more byte efficient due to direct bit addressing and program counter relative branching:
- faster running because fewer bytes of instruction need to be fetched and fewer conditional jumps are processed:
- lower cost because of the high level of system-integration within one component.

These rather unabashed claims of excellence shall not go unsubstantiated. The rest of this chapter examines less trivial tasks simplified by the Boolean processor. The first three compare the 8051 with other micro-processors; the last two go into 8051-based system designs in much greater depth.

Design Example # 1—Bit Permutation

First off, we'll use the bit-transfer instructions to permute a lengthy pattern of bits.



A steadily increasing number of data communication products use encoding methods to protect the security of sensitive information. By law, interstate financial transactions involving the Federal banking system must be transmitted using the Federal Information Processing Data Encryption Standard (DES).

Basically, the DES combines eight bytes of "plaintext" data (in binary, ASCII, or any other format) with a 56-bit "key", producing a 64-bit encrypted value for transmission. At the receiving end the same algorithm is applied to the incoming data using the same key, reproducing the original eight byte message. The algorithm used for these permutations is fixed; different user-defined keys ensure data privacy.

It is not the purpose of this note to describe the DES in any detail. Suffice it to say that encryption/decryption is a long, iterative process consisting of rotations, exclusive -OR operations, function table look-ups, and an extensive (and quite bizarre) sequence of bit permutation, packing, and unpacking steps. (For further details refer to the June 21, 1979 issue of *Electronics* magazine.) The bit manipulation steps are included, it is rumored, to impede a general purpose digital supercomputer trying to "break" the code. Any algorithm implementing the DES with previous generation microprocessors would spend virtually all of its time diddling bits.

The bit manipulation performed is typified by the Key Schedule Calculation represented in Figure 9. This step is repeated 16 times for each key used in the course of a transmission. In essence, a seven-byte, 56-bit "Shifted Key Buffer" is transformed into an eight-byte, "Permutation Buffer" without altering the shifted Key. The arrows in Figure 9 indicate a few of the translation steps. Only six bits of each byte of the Permutation Buffer are used; the two high-order bits of each byte are cleared. This means only 48 of the 56 Shifted Key Buffer bits are used in any one iteration.

Different microprocessor architectures would best implement this type of permutation in different ways. Most approaches would share the steps of Figure 10a:

- Initialize the Permutation Buffer to default state (ones or zeroes):
- Isolate the state of a bit of a byte from the Key Buffer. Depending on the CPU, this might be accomplished by rotating a word of the Key Buffer through a carry flag or testing a bit in memory or an accumulator against a mask byte:
- Perform a conditional jump based on the carry or zero flag if the Permutation Buffer default state is correct:
- Otherwise reverse the corresponding bit in the permutation buffer with logical operations and mask bytes.

Each step above may require several instructions. The last three steps must be repeated for all 48 bits. Most microprocessors would spend 300 to 3,000 microseconds on each of the 16 iterations.

Notice, though, that this flow chart looks a lot like Figure 8. The Boolean Processor can permute bits by simply moving them from the source to the carry to the destination—a total of two instructions taking four bytes and three microseconds per bit. Assume the Shifted Key Buffer and Permutation Buffer both reside in bit-addressable RAM, with the bits of the former assigned symbolic names SKB_1, SKB_2, ... SKB_56, and that the bytes of the latter are named PB_1, ... PB_8. Then working from Figure 9, the software for the permutation algorithm would be that of Example 1a. The total routine length would be 192 bytes, requiring 144 microseconds.

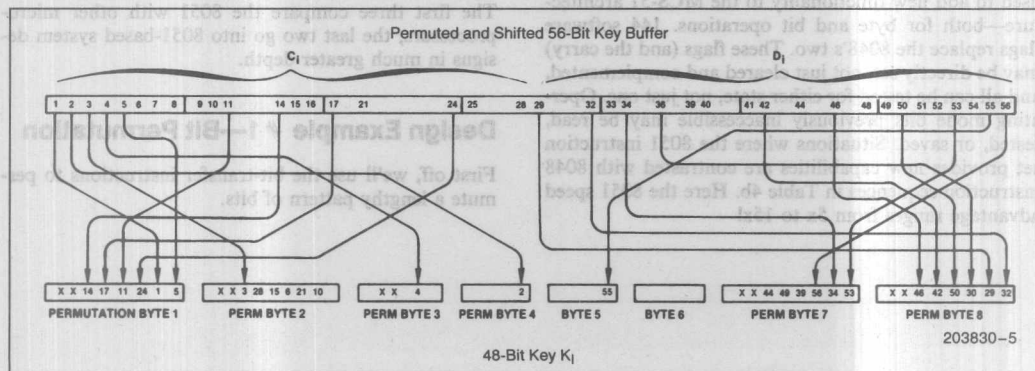


Figure 9. DES Key Schedule Transformation

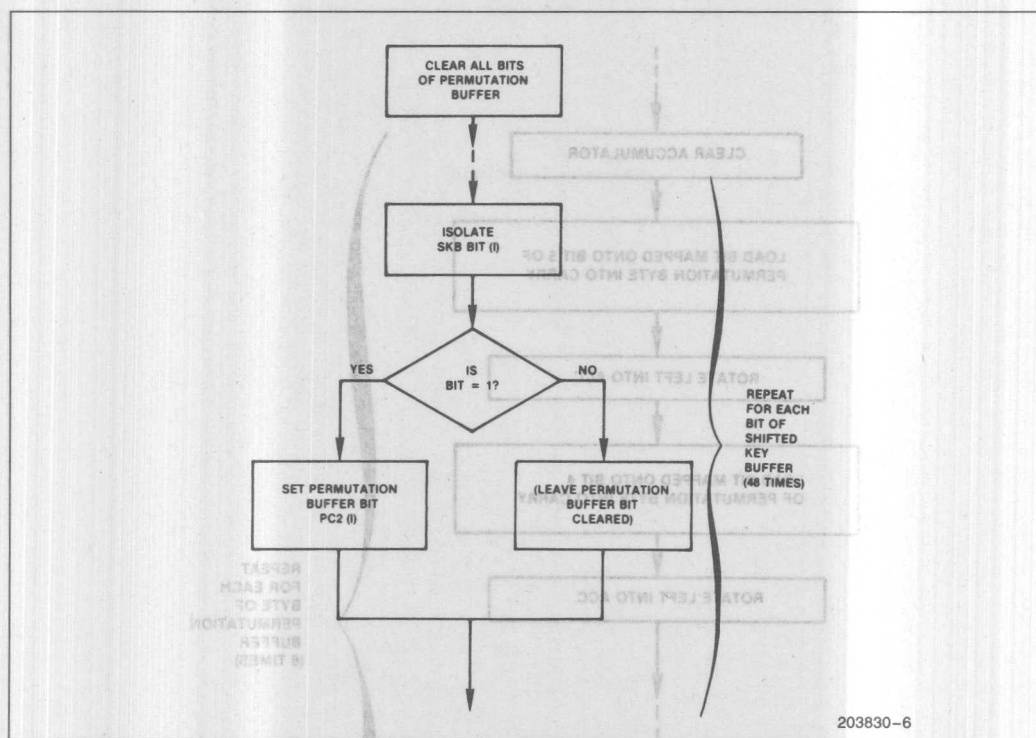
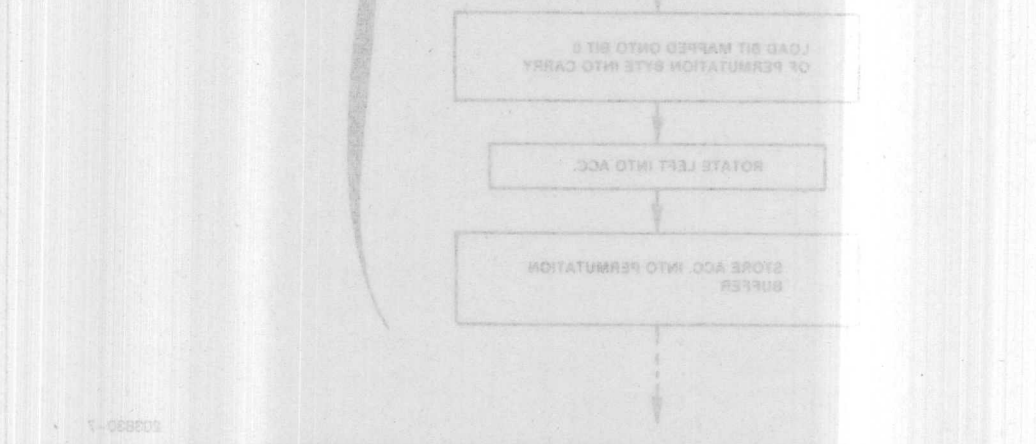


Figure 10a. Flowchart for Key Permutation Attempted with a Byte Processor



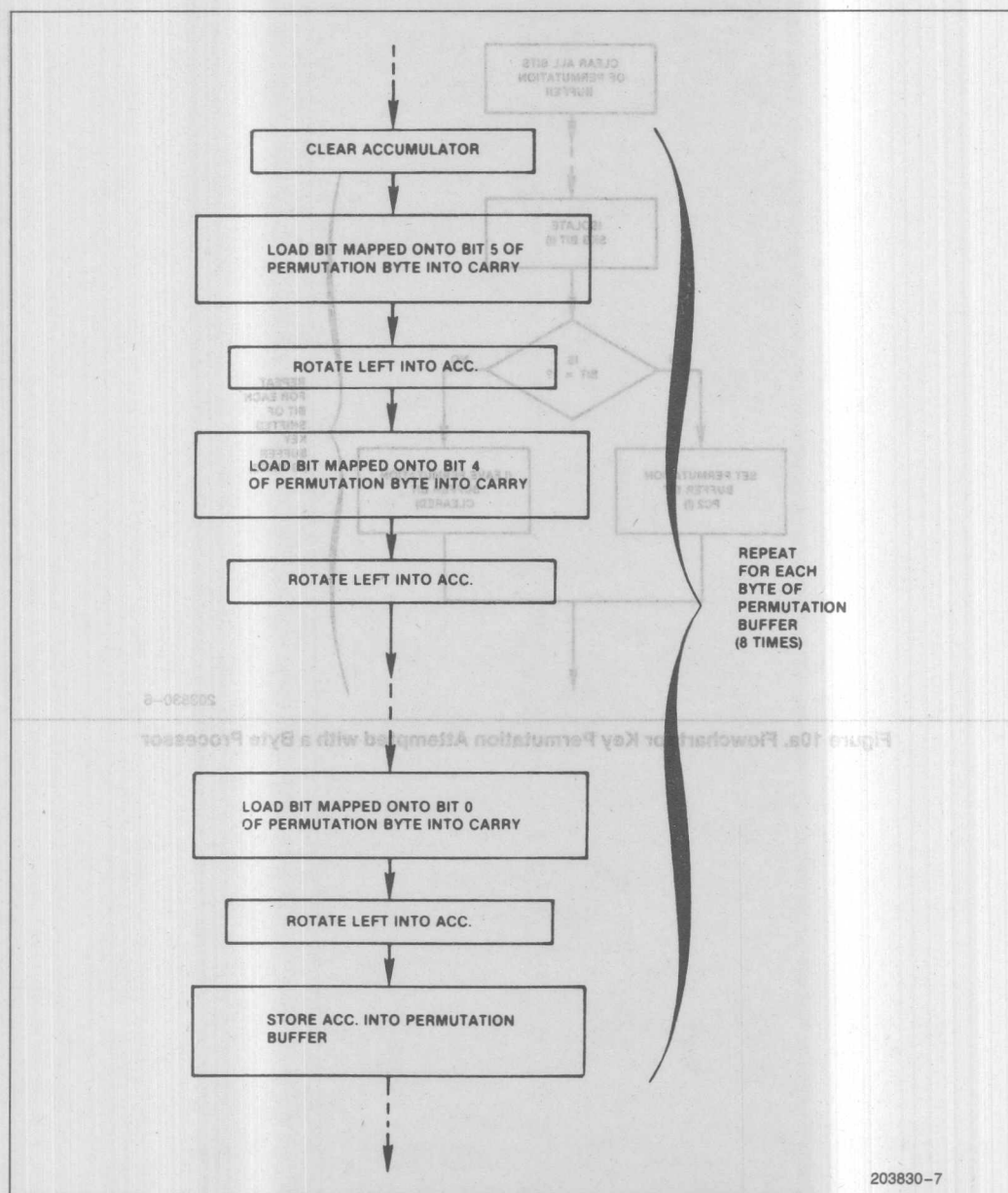


Figure 10b. DES Key Permutation with Boolean Processor

The algorithm of Figure 100 is just slightly more efficient in this time-critical application and illustrates the synergy of an integrated byte and bit processor. The bits needed for each byte of the Permutation Buffer are assimilated by loading each bit into the carry (1 μ s.) and shifting it into the accumulator (1 μ s.). Each byte is stored in RAM when completed. Forty-eight bits thus need a total of 112 instructions, some of which are listed in Example 1b.

Worst-case execution time would be 112 microseconds, since each instruction takes a single cycle. Routine length would also decrease, to 168 bytes. (Actually, in the context of the complete encryption algorithm, each permuted byte would be processed as soon as it is assimilated—saving memory and cutting execution time by another 8 μ s.)

To date, most banking terminals and other systems using the DES have needed special boards or peripheral controller chips just for the encryption/decryption process, and still more hardware to form a serial bit stream for transmission (Figure 11a). An 8051 solution could pack most of the entire system onto the one chip (Figure 11b). The whole DES algorithm would require less than one-fourth of the on-chip program memory, with the remaining bytes free for operating the banking terminal (or whatever) itself.

Moreover, since transmission and reception of data is performed through the on-board UART, the unencrypted data (plaintext) never even exists outside the microcomputer! Naturally, this would afford a high degree of security from data interception.

a.) "Brute Force" technique

```
MOV C,SKB_1
MOV PB_1.1,C
MOV C,SKB_2
MOV PB_4.0,C
MOV C,SKB_3
MOV PB_2.5,C
MOV C,SKB_4
MOV PB_1.0,C
...
...
MOV C,SKB_55
MOV PB_5.0,C
MOV C,SKB_56
MOV PB_7.2,C
```

b.) Using Accumulator to Collect Bits

```
CLR A
MOV C,SKB_14
RLC A
MOV C,SKB_17
RLC A
MOV C,SKB_11
RLC A
MOV C,SKB_24
RLC A
MOV C,SKB_1
RLC A
MOV C,SKB_5
RLC A
MOV PB_1,A
...
...
MOV C,SKB_29
RLC A
MOV C,SKB_32
RLC A
MOV PB_8,A
```

Figure 11a and 11b show algorithms for receiving or transmitting a byte of data. (Another section of program would invoke this algorithm eight times, synchronizing it with a start bit, clock signal, software delay, or timer interrupt.) Data is received by testing an input pin, setting the carry to the same state, shifting the carry into a data buffer, and saving the partial frame in internal RAM. Data is transmitted by shifting an output buffer through the carry, and generating each bit on an output pin.

A side-by-side comparison of the software for this common "bit-banging" application with three different microprocessor architectures is shown in Table 2a and 2b. The 8051 solution is more efficient than the others on every count.

Design Example 4—Software Serial I/O

A circuit often needed on beginning microcomputers is to write a program simulating a UART. Though being the 8051 Family may appear to be a moot point (given that the hardware for a full UART is on-chip), it is still instructive to see how it would be done and maintains a product line tradition.

As it turns out, the 8051 microcomputers can receive or transmit serial data via software very efficiently using the bootstrap instructions. Since any I/O pin may be a serial input or output, several serial links could be maintained at once.

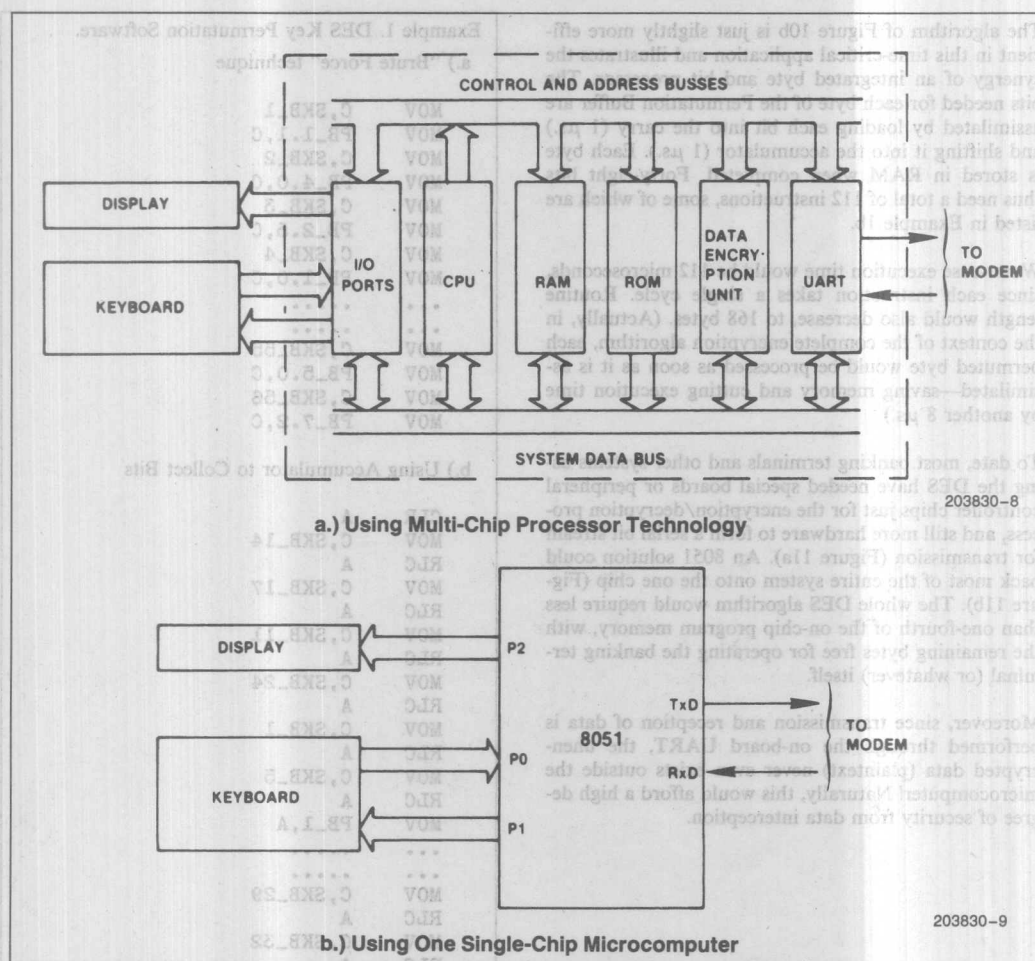


Figure 11. Secure Banking Terminal Block Diagram

Design Example #2—Software Serial I/O

An exercise often imposed on beginning microcomputer students is to write a program simulating a UART. Though doing this with the 8051 Family may appear to be a moot point (given that the hardware for a full UART is on-chip), it is still instructive to see how it would be done, and maintains a product line tradition.

As it turns out, the 8051 microcomputers can receive or transmit serial data via software very efficiently using the Boolean instruction set. Since any I/O pin may be a serial input or output, several serial links could be maintained at once.

Figures 12a and 12b show algorithms for receiving or transmitting a byte of data. (Another section of program would invoke this algorithm eight times, synchronizing it with a start bit, clock signal, software delay, or timer interrupt.) Data is received by testing an input pin, setting the carry to the same state, shifting the carry into a data buffer, and saving the partial frame in internal RAM. Data is transmitted by shifting an output buffer through the carry, and generating each bit on an output pin.

A side-by-side comparison of the software for this common "bit-banging" application with three different microprocessor architectures is shown in Table 5a and 5b. The 8051 solution is more efficient than the others on every count!

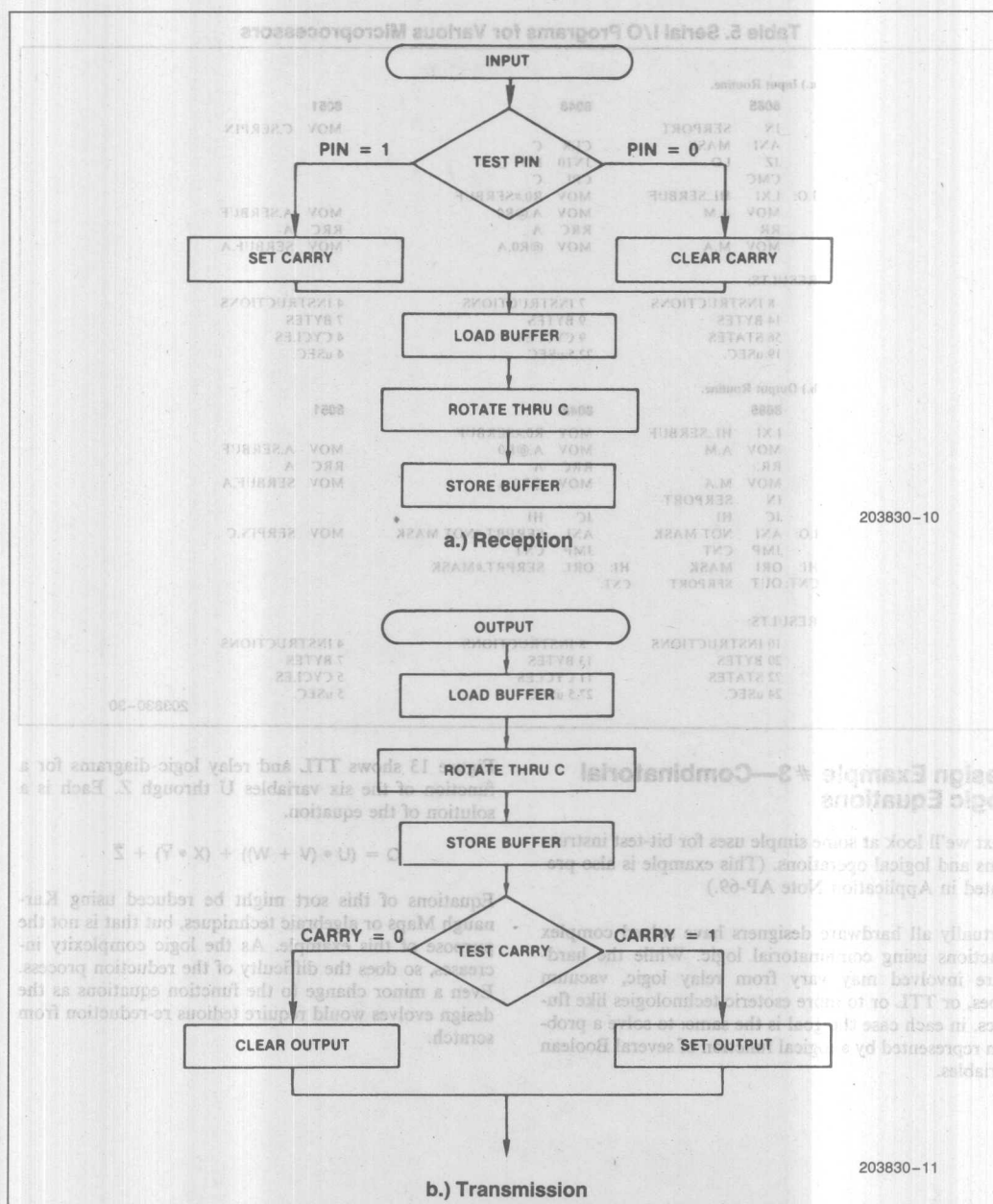


Figure 12. Serial I/O Algorithms

Table 5. Serial I/O Programs for Various Microprocessors

a.) Input Routine.		
8085	8048	8051
IN SERPORT ANI MASK = 01H JZ LO CMC I.O: LXI HL, SERBUF MOV A, M RR MOV M, A	CLR C JNT0 LO CPL C MOV R0, #SERBUF MOV A, @R0 RRC A MOV @R0, A	MOV C, SERPIN I = 01H MOV A, SERBUF RRC A MOV SERBUF, A
RESULTS: 8 INSTRUCTIONS 14 BYTES 56 STATES 19 uSEC.	7 INSTRUCTIONS 9 BYTES 9 CYCLES 22.5 uSEC.	4 INSTRUCTIONS 7 BYTES 4 CYCLES 4 uSEC.
b.) Output Routine.		
8085	8048	8051
LXI HL, SERBUF MOV A, M RR MOV M, A IN SERPORT JC HI I.O: ANI NOT MASK JMP CNT HI: ORI MASK CNT: OUT SERPORT	MOV R0, #SERBUF MOV A, @R0 RRC A MOV @R0, A JC HI ANI SERPRT, #NOT MASK JMP CNT HI: ORI SERPRT, #MASK CNT:	MOV A, SERBUF RRC A MOV SERBUF, A MOV SERPIN, C
RESULTS: 10 INSTRUCTIONS 20 BYTES 72 STATES 24 uSEC.	8 INSTRUCTIONS 13 BYTES 11 CYCLES 27.5 uSEC.	4 INSTRUCTIONS 7 BYTES 5 CYCLES 5 uSEC.

Design Example #3—Combinatorial Logic Equations

Next we'll look at some simple uses for bit-test instructions and logical operations. (This example is also presented in Application Note AP-69.)

Virtually all hardware designers have solved complex functions using combinatorial logic. While the hardware involved may vary from relay logic, vacuum tubes, or TTL or to more esoteric technologies like fluidics, in each case the goal is the same: to solve a problem represented by a logical function of several Boolean variables.

Figure 13 shows TTL and relay logic diagrams for a function of the six variables U through Z. Each is a solution of the equation.

$$Q = (U \cdot (V + W)) + (X \cdot \bar{Y}) + \bar{Z}$$

Equations of this sort might be reduced using Karnaugh Maps or algebraic techniques, but that is not the purpose of this example. As the logic complexity increases, so does the difficulty of the reduction process. Even a minor change to the function equations as the design evolves would require tedious re-reduction from scratch.

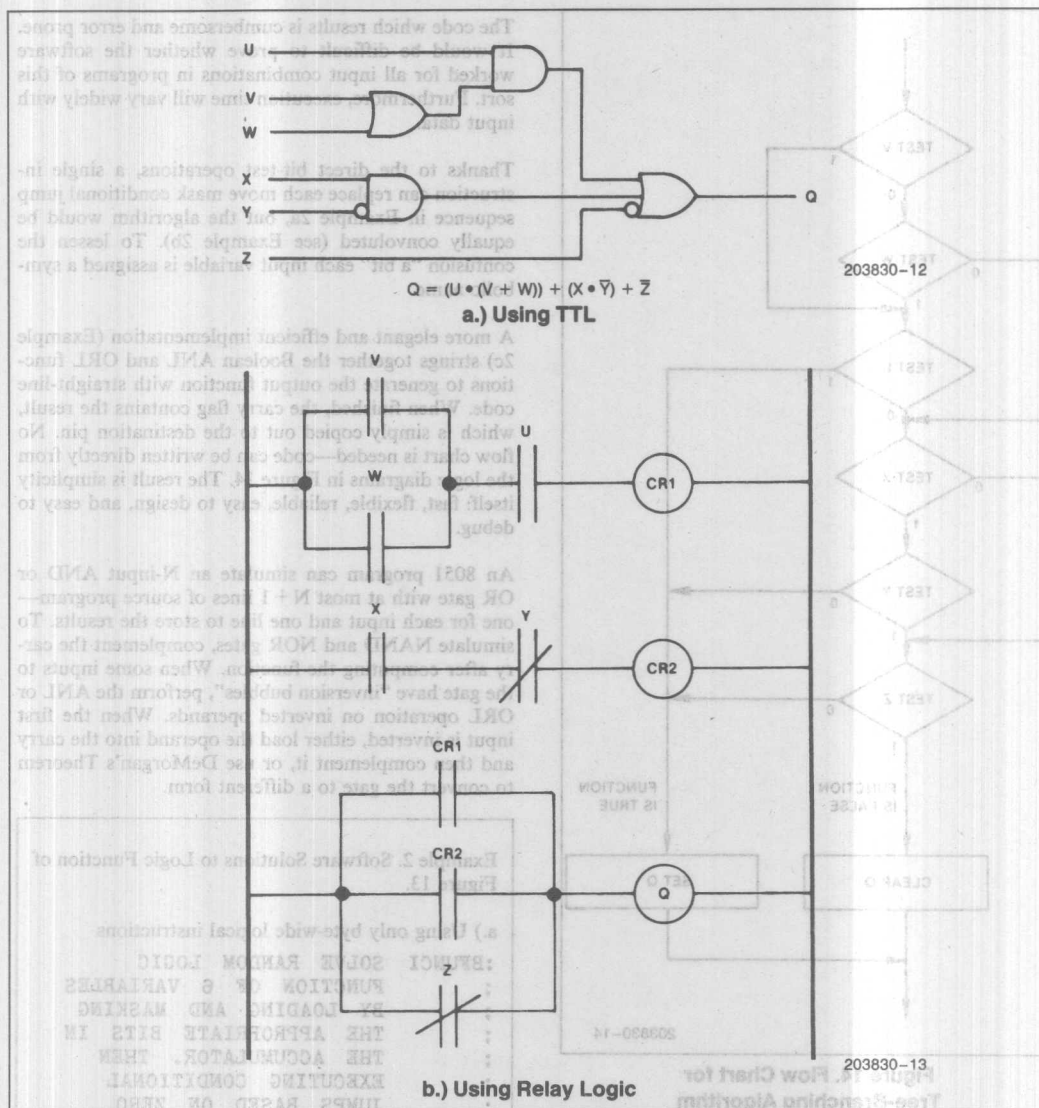


Figure 13. Hardware Implementations of Boolean Functions

For the sake of comparison we will implement this function three ways, restricting the software to three proper subsets of the MCS-51 instruction set. We will also assume that U and V are input pins from different input ports, W and X are status bits for two peripheral controllers, and Y and Z are software flags set up earlier in the program. The end result must be written

to an output pin on some third port. The first two implementations follow the flow-chart shown in Figure 14. Program flow would embark on a route down a test-and-branch tree and leaves either the "True" or "Not True" exit ASAP—as soon as the proper result has been determined. These exits then rewrite the output port with the result bit respectively one or zero.

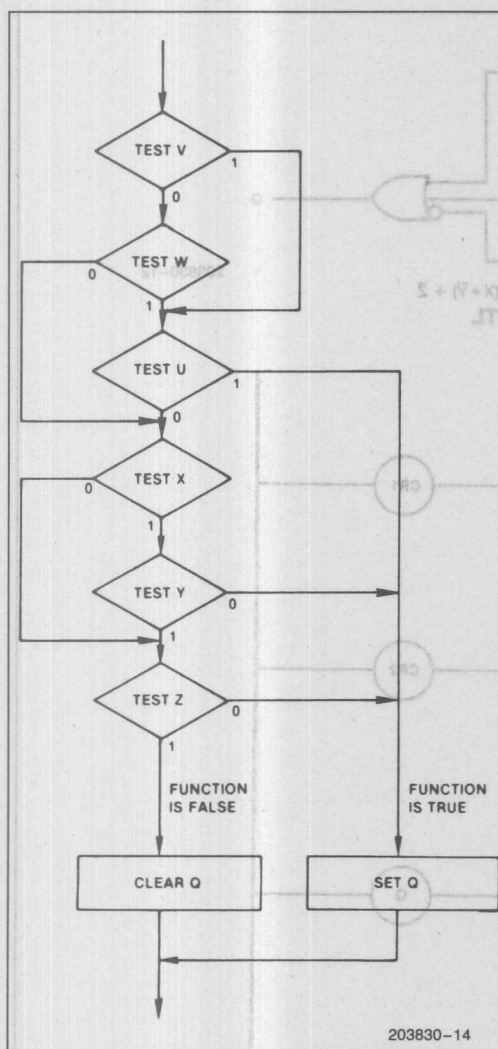


Figure 14. Flow Chart for Tree-Branching Algorithm

Other digital computers must solve equations of this type with standard word-wide logical instructions and conditional jumps. So for the first implementation, we won't use any generalized bit-addressing instructions. As we shall soon see, being constrained to such an instruction subset produces somewhat sloppy software solutions. MCS-51 mnemonics are used in Example 2a: other machines might further cloud the situation by requiring operation-specific mnemonics like INPUT, OUTPUT, LOAD, STORE, etc., instead of the MOV mnemonic used for all variable transfers in the 8051 instruction set.

The code which results is cumbersome and error prone. It would be difficult to prove whether the software worked for all input combinations in programs of this sort. Furthermore, execution time will vary widely with input data.

Thanks to the direct bit-test operations, a single instruction can replace each move mask conditional jump sequence in Example 2a, but the algorithm would be equally convoluted (see Example 2b). To lessen the confusion "a bit" each input variable is assigned a symbolic name.

A more elegant and efficient implementation (Example 2c) strings together the Boolean ANL and ORL functions to generate the output function with straight-line code. When finished, the carry flag contains the result, which is simply copied out to the destination pin. No flow chart is needed—code can be written directly from the logic diagrams in Figure 14. The result is simplicity itself: fast, flexible, reliable, easy to design, and easy to debug.

An 8051 program can simulate an N-input AND or OR gate with at most N + 1 lines of source program—one for each input and one line to store the results. To simulate NAND and NOR gates, complement the carry after computing the function. When some inputs to the gate have "inversion bubbles", perform the ANL or ORL operation on inverted operands. When the first input is inverted, either load the operand into the carry and then complement it, or use DeMorgan's Theorem to convert the gate to a different form.

Example 2. Software Solutions to Logic Function of Figure 13.

a.) Using only byte-wide logical instructions

```
:BFUNCI SOLVE RANDOM LOGIC
;      FUNCTION OF 6 VARIABLES
;      BY LOADING AND MASKING
;      THE APPROPRIATE BITS IN
;      THE ACCUMULATOR. THEN
;      EXECUTING CONDITIONAL
;      JUMPS BASED ON ZERO
;      CONDITION. (APPROACH USED
;      BY BYTE-ORIENTED
;      ARCHITECTURES.) BYTE AND
;      MASK VALUES CORRESPOND TO
;      RESPECTIVE BYTE ADDRESS
;      AND BIT POSITIONS.
OUTBUF DATA 22H
;OUTPUT PIN STATE MAP
;
```

b.) Using only bit-test instructions

c.) Using logical operations on Boolean variables

```
MOV C,V  
ORL C,W ;OUTPUT OF OR GATE  
ANL C,U ;OUTPUT OF TOP AND GATE  
MOV FO,C ;SAVE INTERMEDIATE STATE  
MOV C,X  
ANL C,Y ;OUTPUT OF BOTTOM AND GATE  
ORL C,FO ;INCLUDE VALUE SAVED ABOVE  
ORL C,Z ;INCLUDE LAST INPUT  
;VARIABLE  
MOV Q,C ;OUTPUT COMPUTED RESULT
```

An upper-limit can be placed on the complexity of software to simulate a large number of gates by summing the total number of inputs and outputs. The *actual* total should be somewhat shorter, since calculations can be "chained," as shown. The output of one gate is often the first input to another, bypassing the intermediate variable to eliminate two lines of source.

Design Example #4—Automotive Dashboard Functions

Now let's apply these techniques to designing the software for a complete controller system. This application is patterned after a familiar real-world application which isn't nearly as trivial as it might first appear: automobile turn signals.

Imagine the three position turn lever on the steering column as a single-pole, triple-throw toggle switch. In its central position all contacts are open. In the up or down positions contacts close causing corresponding lights in the rear of the car to blink. So far very simple.

Two more turn signals blink in the front of the car, and two others in the dashboard. All six bulbs flash when an emergency switch is closed. A thermo-mechanical relay (accessible under the dashboard in case it wears out) causes the blinking.

Applying the brake pedal turns the tail light filaments on constantly . . . unless a turn is in progress, in which case the blinking tail light is not affected. (Of course, the front turn signals and dashboard indicators are not affected by the brake pedal.) Table 6 summarizes these operating modes.

Table 6. Truth Table for Turn-Signal Operation

Input Signals				Output Signals			
Brake Switch	Emerg. Switch	Left Turn Switch	Right Turn Switch	Left Front & Dash	Right Front & Dash	Left Rear	Right Rear
0	0	0	0	Off	Off	Off	Off
0	0	0	1	Off	Blink	Off	Blink
0	0	1	0	Blink	Off	Blink	Off
0	1	0	0	Blink	Blink	Blink	Blink
0	1	0	1	Blink	Blink	Blink	Blink
1	0	0	0	Off	Off	On	On
1	0	0	1	Off	Blink	On	Blink
1	0	1	0	Blink	Off	Blink	On
1	1	0	0	Blink	Blink	On	On
1	1	0	1	Blink	Blink	On	Blink
1	1	1	0	Blink	Blink	Blink	On
1	1	1	1	Blink	Blink	Blink	On

But we're not done yet. Each of the exterior turn signal (but not the dashboard) bulbs has a second, somewhat dimmer filament for the parking lights. Figure 15 shows TTL circuitry which could control all six bulbs. The signals labeled "High Freq." and "Low Freq." represent two square-wave inputs. Basically, when one of the turn switches is closed or the emergency switch is activated the low frequency signal (about 1 Hz) is gated through to the appropriate dashboard indicator(s) and turn signal(s). The rear signals are also activated when the brake pedal is depressed provided a turn is not being made in the same direction. When the parking light switch is closed the higher frequency oscillator is gated to each front and rear turn signal, sustaining a low-intensity background level. (This is to eliminate the need for additional parking light filaments.)

In most cars, the switching logic to generate these functions requires a number of multiple-throw contacts. As many as 18 conductors thread the steering column of some automobiles solely for turn-signal and emergency blinker functions. (The author discovered this recently to his astonishment and dismay when replacing the whole assembly because of one burned contact.)

A multiple-conductor wiring harness runs to each corner of the car, behind the dash, up the steering column, and down to the blinker relay below. Connectors at

each termination for each filament lead to extra cost and labor during construction, lower reliability and safety, and more costly repairs. And considering the system's present complexity, increasing its reliability or detecting failures would be quite difficult.

There are two reasons for going into such painful detail describing this example. First, to show that the messiest part of many system designs is determining what the controller should do. Writing the software to solve these functions will be comparatively easy. Secondly, to show the many potential failure points in the system. Later we'll see how the peripheral functions and intelligence built into a microcomputer (with a little creativity) can greatly reduce external interconnections and mechanical part count.

The Single-Chip Solution

The circuit shown in Figure 16 indicates five input pins to the five input variables—left-turn select, right-turn select, brake pedal down, emergency switch on, and parking lights on. Six output pins turn on the front, rear, and dashboard indicators for each side. The microcomputer implements all logical functions through software, which periodically updates the output signals as time elapses and input conditions change.

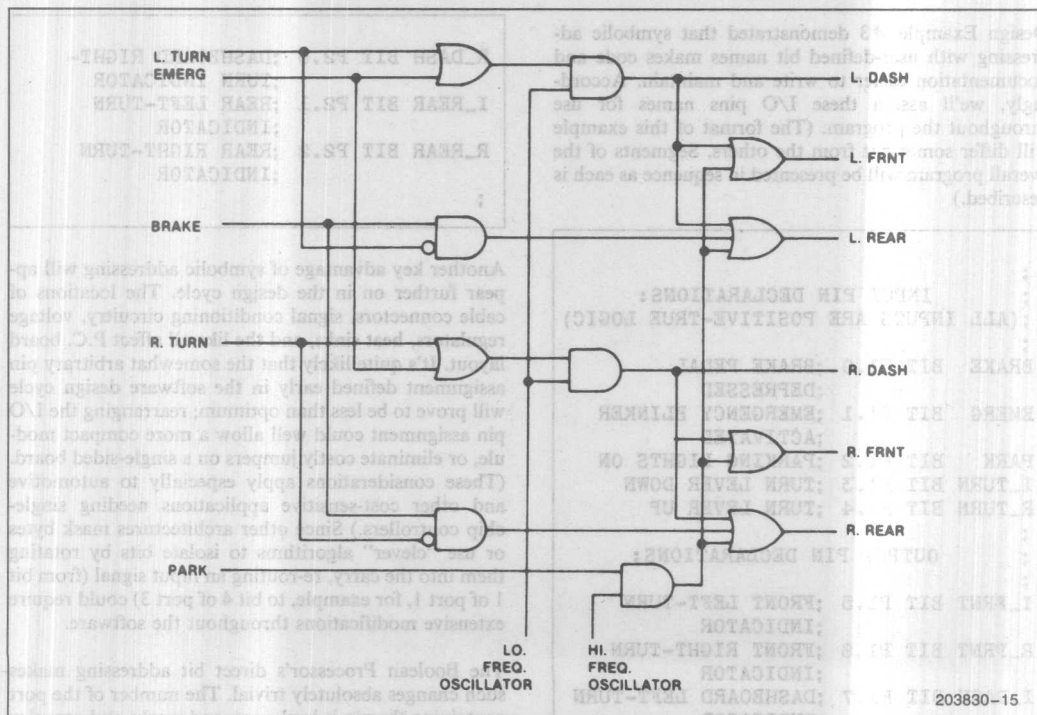


Figure 15. TTL Logic Implementation of Automotive Turn Signals

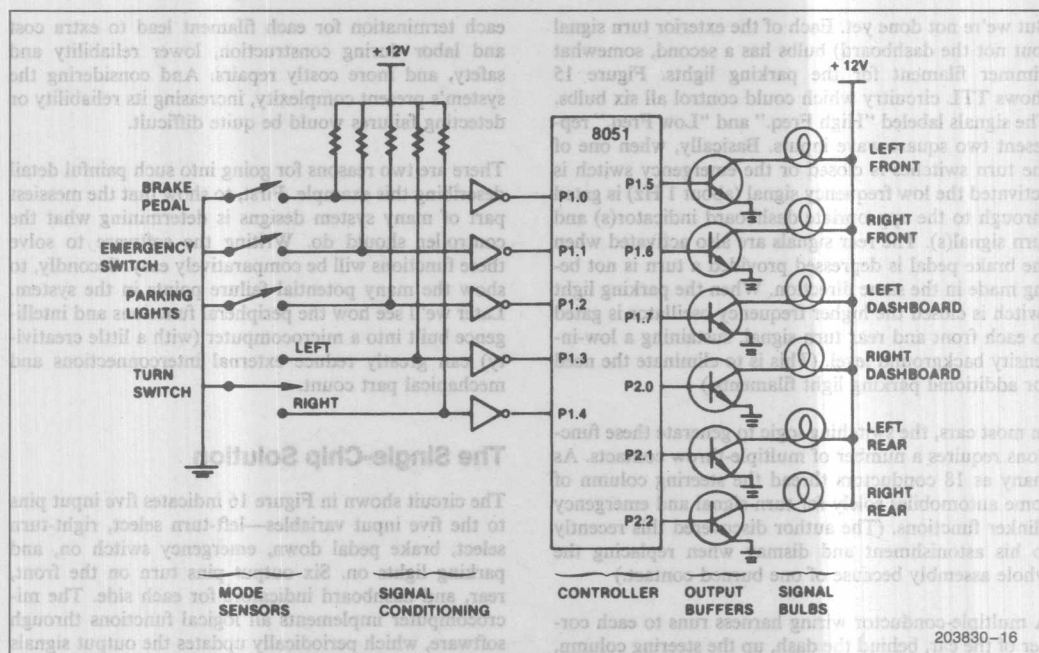


Figure 16. Microcomputer Turn-Signal Connections

Design Example #3 demonstrated that symbolic addressing with user-defined bit names makes code and documentation easier to write and maintain. Accordingly, we'll assign these I/O pins names for use throughout the program. (The format of this example will differ somewhat from the others. Segments of the overall program will be presented in sequence as each is described.)

```

;
; INPUT PIN DECLARATIONS:
;(ALL INPUTS ARE POSITIVE-TRUE LOGIC)
;
BRAKE BIT P1.0 ;BRAKE PEDAL
;DEPRESSED
EMERG BIT P1.1 ;EMERGENCY BLINKER
;ACTIVATED
PARK BIT P1.2 ;PARKING LIGHTS ON
I_TURN BIT P1.3 ;TURN LEVER DOWN
R_TURN BIT P1.4 ;TURN LEVER UP
;
; OUTPUT PIN DECLARATIONS:
;
I_FRNT BIT P1.5 ;FRONT LEFT-TURN
;INDICATOR
R_FRNT BIT P1.6 ;FRONT RIGHT-TURN
;INDICATOR
I_DASH BIT P1.7 ;DASHBOARD LEFT-TURN
;INDICATOR

```

```

R_DASH BIT P2.0 ;DASHBOARD RIGHT-
;TURN INDICATOR
I_REAR BIT P2.1 ;REAR LEFT-TURN
;INDICATOR
R_REAR BIT P2.2 ;REAR RIGHT-TURN
;INDICATOR
;

```

Another key advantage of symbolic addressing will appear further on in the design cycle. The locations of cable connectors, signal conditioning circuitry, voltage regulators, heat sinks, and the like all affect P.C. board layout. It's quite likely that the somewhat arbitrary pin assignment defined early in the software design cycle will prove to be less than optimum; rearranging the I/O pin assignment could well allow a more compact module, or eliminate costly jumpers on a single-sided board. (These considerations apply especially to automotive and other cost-sensitive applications needing single-chip controllers.) Since other architectures mask bytes or use "clever" algorithms to isolate bits by rotating them into the carry, re-routing an input signal (from bit 1 of port 1, for example, to bit 4 of port 3) could require extensive modifications throughout the software.

The Boolean Processor's direct bit addressing makes such changes absolutely trivial. The number of the port containing the pin is irrelevant, and masks and complex

program structures are not needed. Only the initial Boolean variable declarations need to be changed; ASM51 automatically adjusts all addresses and symbolic references to the reassigned variables. The user is assured that no additional debugging or software verification will be required.

```

; *****
; INTERRUPT RATE SUBDIVIDER
SUB_DIV DATA 20H
; HIGH-FREQUENCY OSCILLATOR BIT
HI_FREQ BIT SUB_DIV, 0
; LOW-FREQUENCY OSCILLATOR BIT
LO_FREQ BIT SUB_DIV, 7
;
; ORG 0000H
JMP INIT
; ORG 100H
; PUT TIMER 0 IN MODE 1
INIT: MOV TMOD, #00000001B
; INITIALIZE TIMER REGISTERS
MOV TLO, #0
MOV TH0, #-16
; SUBDIVIDE INTERRUPT RATE BY 244
MOV SUB_DIV, #244
; ENABLE TIMER INTERRUPTS
SETB ETO
; GLOBALLY ENABLE ALL INTERRUPTS
SETB EA
; START TIMER
SETB TRO
;
; (CONTINUE WITH BACKGROUND PROGRAM)
;
; PUT TIMER 0 IN MODE 1
; INITIALIZE TIMER REGISTERS

; SUBDIVIDE INTERRUPT RATE BY 244
; ENABLE TIMER INTERRUPTS
; GLOBALLY ENABLE ALL INTERRUPTS
; START TIMER

```

Timer 0 (one of the two on-chip timer counters) replaces the thermo-mechanical blinker relay in the dashboard controller. During system initialization it is configured as a timer in mode 1 by setting the least significant bit of the timer mode register (TMOD). In this configuration the low-order byte (TLO) is incremented every machine cycle, overflowing and incrementing the high-order byte (TH0) every 256 μ s. Timer interrupt 0 is enabled so that a hardware interrupt will occur each time TH0 overflows.

An eight-bit variable in the bit-addressable RAM array will be needed to further subdivide the interrupts via software. The lowest-order bit of this counter toggles very fast to modulate the parking lights: bit 7 will be

“tuned” to approximately 1 Hz for the turn- and emergency-indicator blinking rate.

Loading TH0 with -16 will cause an interrupt after 4.096 ms. The interrupt service routine reloads the high-order byte of timer 0 for the next interval, saves the CPU registers likely to be affected on the stack, and then decrements SUB_DIV. Loading SUB_DIV with 244 initially and each time it decrements to zero will produce a 0.999 second period for the highest-order bit.

```

ORG 000BH ; TIMER 0 SERVICE VECTOR
MOV TH0, #-16
PUSH PSW
PUSH ACC
PUSH B
DJNZ SUB_DIV, TOSERV
MOV SUB_DIV, #244

```

The code to sample inputs, perform calculations, and update outputs—the real “meat” of the signal controller algorithm—may be performed either as part of the interrupt service routine or as part of a background program loop. The only concern is that it must be executed at least several dozen times per second to prevent parking light flickering. We will assume the former case, and insert the code into the timer 0 service routine.

First, notice from the logic diagram (Figure 15) that the subterm (PARK • H_FREQ), asserted when the parking lights are to be on dimly, figures into four of the six output functions. Accordingly, we will first compute that term and save it in a temporary location named “DIM”. The PSW contains two general purpose flags: F0, which corresponds to the 8048 flag of the same name, and PSW.1. Since the PSW has been saved and will be restored to its previous state after servicing the interrupt, we can use either bit for temporary storage.

```

DIM BIT PSW.1 ; DECLARE TEMP
; STORAGE FLAG
; ...
MOV C, PARK ; GATE PARKING
; LIGHT SWITCH
ANL HI_FREQ ; WITH HIGH
; FREQUENCY
; SIGNAL
MOV DIM, C ; AND SAVE IN
; TEMP. VARIABLE

```

This simple three-line section of code illustrates a remarkable point. The software indicates in very abstract terms exactly what function is being performed, inde-

these three bits include an input pin, a bit within a program variable, and a software flag in the PSW is totally invisible to the programmer.

Now generate and output the dashboard left turn signal.

```

;
MOV C,L_TURN      ;SET CARRY IF
                  ;TURN
ORL C,EMERG        ;OR EMERGENCY
                  ;SELECTED
ANL C,LO_FREQ      ;GATE IN 1 HZ
                  ;SIGNAL
MOV I_DASH,C       ;AND OUTPUT TO
                  ;DASHBOARD

```

To generate the left front turn signal we only need to add the parking light function in FO. But notice that the function in the carry will also be needed for the rear signal. We can save effort later by saving its current state in FO.

```

;
MOV FO,C           ;SAVE FUNCTION
                  ;SO FAR
ORL C,DIM           ;ADD IN PARKING
                  ;LIGHT FUNCTION
MOV L_FRNT,C       ;AND OUTPUT TO
                  ;TURN SIGNAL

```

Finally, the rear left turn signal should also be on when the brake pedal is depressed, provided a left turn is not in progress.

```

MOV C,BRAKE        ;GATE BRAKE
                  ;PEDAL SWITCH
ANL C,L_TURN       ;WITH TURN
                  ;LEVER
ORL C,FO            ;INCLUDE TEMP.
                  ;VARIABLE FROM DASH

```

```

ORL C,DIM           ;AND PARKING
                  ;LIGHT FUNCTION
MOV L_REAR,C       ;AND OUTPUT TO
                  ;TURN SIGNAL

```

Now we have to go through a similar sequence for the right-hand equivalents to all the left-turn lights. This also gives us a chance to see how the code segments above look when combined.

```

MOV C,R_TURN       ;SET CARRY H-
                  ;TURN
ORL C,EMERG        ;OR EMERGENCY
                  ;SELECTED
ANL C,LO_FREQ      ;IF SO. GATE IN 1
                  ;HZ SIGNAL
MOV R_DASH.C       ;AND OUTPUT TO
                  ;DASHBOARD
MOV FO.C           ;SAVE FUNCTION
                  ;SO FAR
ORL C,DIM           ;ADD IN PARKING
                  ;LIGHT FUNCTION
MOV R_FRNT.C       ;AND OUTPUT TO
                  ;TURN SIGNAL
MOV C,BRAKE        ;GATE BRAKE
                  ;PEDAL SWITCH
ANL C,R_TURN       ;WITH TURN
                  ;LEVER
ORL C,FO            ;INCLUDE TEMP.
                  ;VARIABLE FROM
                  ;DASH
ORL C,DIM           ;AND PARKING
                  ;LIGHT FUNCTION
MOV R_REAR.C       ;AND OUTPUT TO
                  ;TURN SIGNAL

```

(The perceptive reader may notice that simply rearranging the steps could eliminate one instruction from each sequence.)

Now that all six bulbs are in the proper states, we can return from the interrupt routine, and the program is finished. This code essentially needs to reverse the status saving steps at the beginning of the interrupt.

Table 7. Non-Trivial Duty Cycles

Sub_Div Bits								Duty Cycles						
7	6	5	4	3	2	1	0	12.5%	25.0%	37.5%	50.0%	62.5%	75.0%	87.5%
X	X	X	X	X	0	0	0	Off	Off	Off	Off	Off	Off	Off
X	X	X	X	X	0	0	1	Off	Off	Off	Off	Off	Off	On
X	X	X	X	X	0	1	0	Off	Off	Off	Off	Off	On	On
X	X	X	X	X	0	1	1	Off	Off	Off	Off	On	On	On
X	X	X	X	X	1	0	0	Off	Off	Off	On	On	On	On
X	X	X	X	X	1	0	1	Off	Off	On	On	On	On	On
X	X	X	X	X	1	1	0	Off	On	On	On	On	On	On
X	X	X	X	X	1	1	1	On	On	On	On	On	On	On


```
POP B ;RESTORE CPU
;REGISTERS.
POP ACC
POP PSW
RETI
```

Program Refinements. The luminescence of an incandescent light bulb filament is generally non-linear: the 50% duty cycle of HI_FREQ may not produce the desired intensity. If the application requires, duty cycles of 25%, 75%, etc. are easily achieved by ANDing and ORing in additional low-order bits of SUB_DIV. For example, 30 H/ signals of seven different duty cycles could be produced by considering bits 2-0 as shown in Table 7. The only software change required would be to the code which sets-up variable DIM;

```
MOV C,SUB_DIV.1;START WITH 50
;PERCENT
ANL C,SUB_DIV.0;MASK DOWN TO 25
;PERCENT
ORL C,SUB_DIV.2;AND BUILD BACK TO
;62 PERCENT
MOV DIM,C ;DUTY CYCLE FOR
;PARKING LIGHTS.
```

Interconnections increase cost and decrease reliability. The simple buffered pin-per-function circuit in Figure 16 is insufficient when many outputs require higher-than-TTL drive levels. A lower-cost solution uses the 8051 serial port in the shift-register mode to augment I/O. In mode 0, writing a byte to the serial port data buffer (SBUF) causes the data to be output sequentially through the "RXD" pin while a burst of eight clock pulses is generated on the "TXD" pin. A shift register connected to these pins (Figure 17) will load the data byte as it is shifted out. A number of special peripheral

driver circuits combining shift-register inputs with high drive level outputs have been introduced recently.

Cascading multiple shift registers end-to-end will expand the number of outputs even further. The data rate in the I/O expansion mode is one megabaud, or 8 μ s. per byte. This is the mode which the serial port defaults to following a reset, so no initialization is required.

The software for this technique uses the B register as a "map" corresponding to the different output functions. The program manipulates these bits instead of the output pins. After all functions have been calculated the B register is shifted by the serial port to the shift-register driver. (While some outputs may glitch as data is shifted through them, at 1 Megabaud most people wouldn't notice. Some shift registers provide an "enable" bit to hold the output states while new data is being shifted in.)

This is where the earlier decision to address bits symbolically throughout the program is going to pay off. This major I/O restructuring is nearly as simple to implement as rearranging the input pins. Again, only the bit declarations need to be changed.

```
I_FRNT BIT B.0 ;FRONT LEFT-TURN
;INDICATOR
R_FRNT BIT B.1 ;FRONT RIGHT-TURN
;INDICATOR
I_DASH BIT B.2 ;DASHBOARD LEFT-TURN
;INDICATOR
R_DASH BIT B.3 ;DASHBOARD RIGHT-TURN
;INDICATOR
I_REAR BIT B.4 ;REAR LEFT-TURN
;INDICATOR
R_REAR BIT B.5 ;REAR RIGHT-TURN
;INDICATOR
```

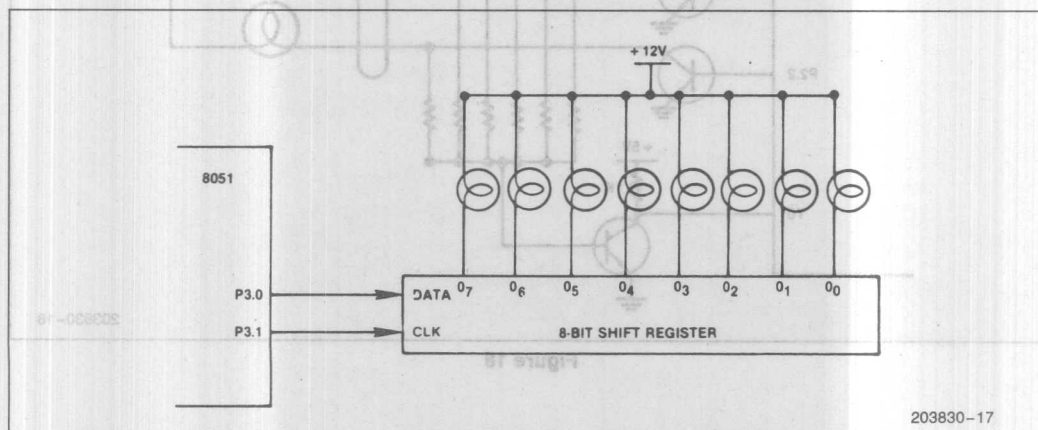


Figure 17. Output Expansion Using Serial Port
2-59

The original program to compute the functions need not change. After computing the output variables, the control map is transmitted to the buffered shift register through the serial port.

MOV SBUF, B ;LOAD BUFFER AND TRANSMIT

The Boolean Processor solution holds a number of advantages over older methods. Fewer switches are required. Each is simpler, requiring fewer poles and lower current contacts. The flasher relay is eliminated entirely. Only six filaments are driven, rather than 10. The wiring harness is therefore simpler and less expensive—one conductor for each of the six lamps and each of the five sensor switches. The fewer conductors use far fewer connectors. The whole system is more reliable.

And since the system is much simpler it would be feasible to implement redundancy and or fault detection on the four main turn indicators. Each could still be a

standard double filament bulb, but with the filaments driven in parallel to tolerate single-element failures.

Even with redundancy, the lights will eventually fail. To handle this inescapable fact current or voltage sensing circuits on each main drive wire can verify that each bulb and its high-current driver is functioning properly. Figure 18 shows one such circuit.

Assume all of the lights are turned on except one; i.e., all but one of the collectors are grounded. For the bulb which is turned off, if there is continuity from +12V through the bulb base and filament, the control wire, all connectors, and the P.C. board traces, and if the transistor is indeed not shorted to ground, then the collector will be pulled to +12V. This turns on the base of Q8 through the corresponding resistor, and grounds the input pin, verifying that the bulb circuit is operational. The continuity of each circuit can be checked by software in this way.

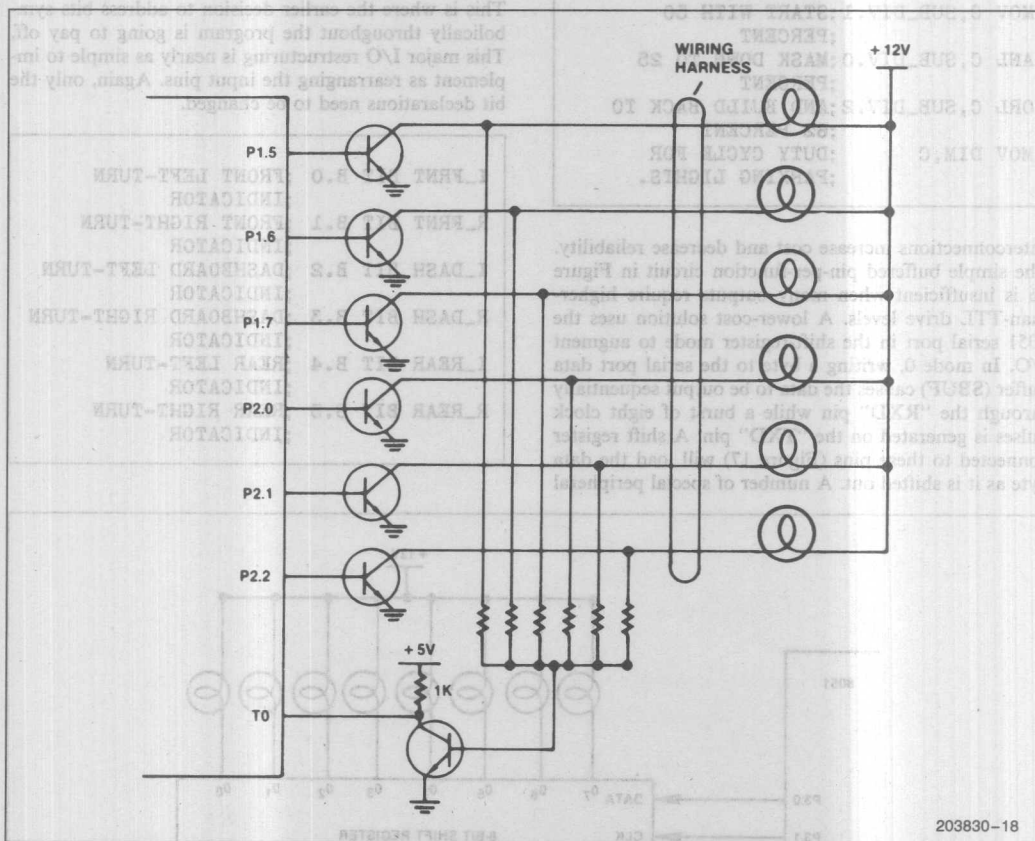


Figure 18

203830-18

Now turn *all* the bulbs on, grounding all the collectors. Q7 should be turned off, and the Test pin should be high. However, a control wire shorted to +12V or an open-circuited drive transistor would leave one of the collectors at the higher voltage even now. This too would turn on Q7, indicating a different type of failure. Software could perform these checks once per second by executing the routine every time the software counter SUB_DIV is reloaded by the interrupt routine.

```

DJNZ SUB_DIV,TOSERV
MOV SUB_DIV,#244      ;RELOAD COUNTER
ORL P1,#11100000B     ;SET CONTROL
                      ;OUTPUTS HIGH

ORL P2,#00000111B     ;FLOAT DRIVE
CLR L_FRNT            ;COLLECTOR

JB TO,FAULT           ;TO SHOULD BE
                      ;PULLED LOW
SETB L_FRNT           ;PULL COLLECTOR
                      ;BACK DOWN

CLR L_DASH
JB TO,FAULT
SETB L_DASH
CLR L_REAR
JB TO,FAULT
SETB L_REAR
CLR R_FRNT
JB TO,FAULT
SETB R_FRNT
CLR R_DASH
JB TO,FAULT
SETB R_DASH
CLR R_REAR
JB TO,FAULT
SETB R_REAR

;
;WITH ALL COLLECTORS GROUNDED. TO
;SHOULD BE HIGH
;IF SO. CONTINUE WITH INTERRUPT
;ROUTINE.
JB TO,TOSERV
FAULT:                ;ELECTRICAL
                      ;FAILURE
                      ;PROCESSING
                      ;ROUTINE
                      ;(LEFT TO
                      ;READER'S
                      ;IMAGINATION)
TOSERV:               ;CONTINUE WITH
                      ;INTERRUPT
                      ;PROCESSING

```

The complete assembled program listing is printed in Appendix A. The resulting code consists of 67 program statements, not counting declarations and comments, which assemble into 150 bytes of object code. Each pass through the service routine requires (coincidentally) 67 μ s plus 32 μ s once per second for the electrical test. If executed every 4 ms as suggested this software would typically reduce the throughput of the background program by less than 2%.

Once a microcomputer has been designed into a system, new features suddenly become virtually free. Software could make the emergency blinkers flash alternately or at a rate faster than the turn signals. Turn signals could override the emergency blinkers. Adding more bulbs would allow multiple tail light sequencing and syncopation—true flash factor, so to speak.

Design Example #5—Complex Control Functions

Finally, we'll mix byte and bit operations to extend the use of 8051 into extremely complex applications.

Programmers can arbitrarily assign I/O pins to input and output functions only if the total does not exceed 32, which is insufficient for applications with a very large number of input variables. One way to expand the number of inputs is with a technique similar to multiplexed-keyboard scanning.

Figure 19 shows a block diagram for a moderately complex programmable industrial controller with the following characteristics:

- 64 input variable sensors:
- 12 output signals:
- Combinational and sequential logic computations:
- Remote operation with communications to a host processor via a high-speed full-duplex serial link:
- Two prioritized external interrupts:
- Internal real-time and time-of-day clocks.

While many microprocessors could be programmed to provide these capabilities with assorted peripheral support chips, an 8051 microcomputer needs no other integrated circuits!

The 64 input sensors are logically arranged as an 8x8 matrix. The pins of Port 1 sequentially enable each column of the sensor matrix: as each is enabled Port 0 reads in the state of each sensor in that column. An eight-byte block in bit-addressable RAM remembers the data as it is read in so that after each complete scan cycle there is an internal map of the current state of all sensors. Logic functions can then directly address the elements of the bit map.

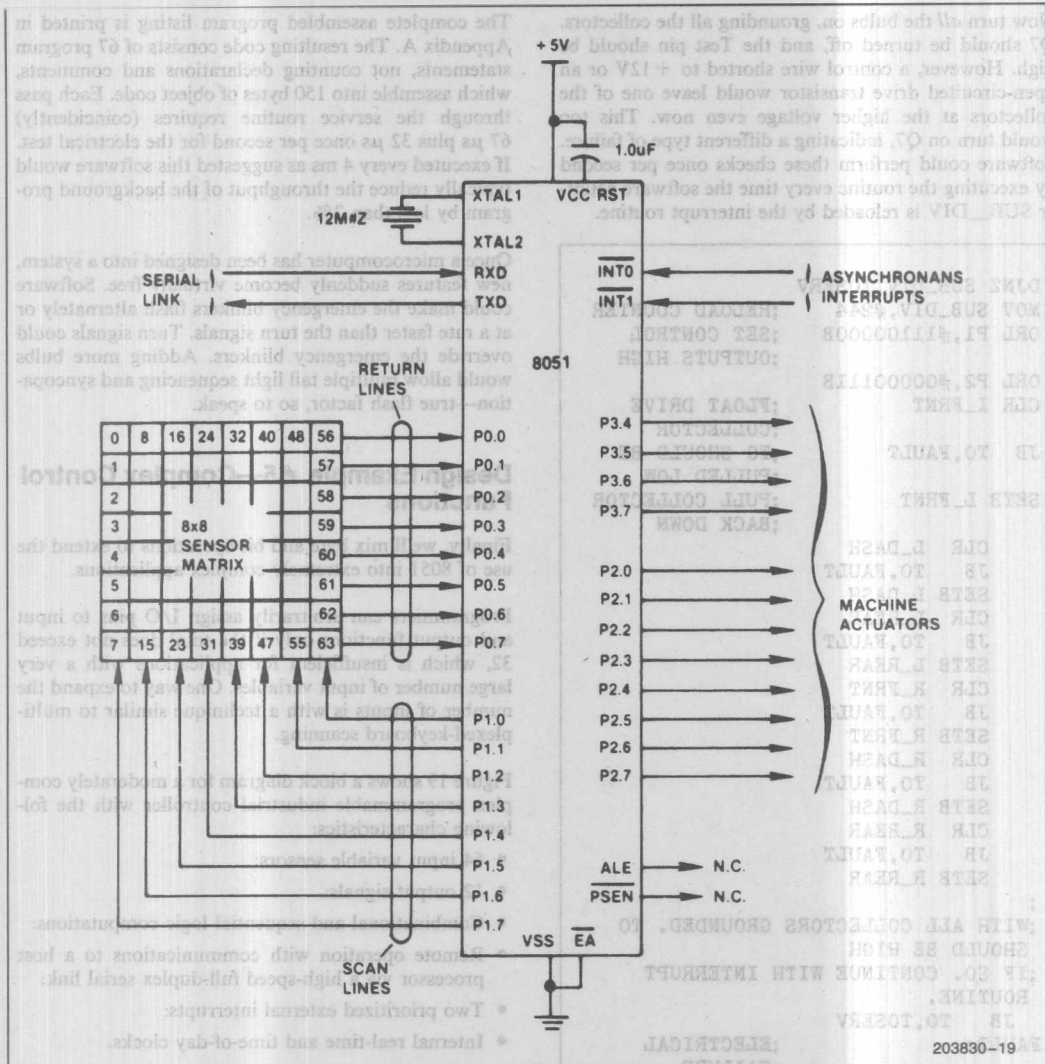


Figure 19. Block Diagram of 64-Input Machine Controller

The computer's serial port is configured as a nine-bit UART, transferring data at 17,000 bytes-per-second. The ninth bit may distinguish between address and data bytes.

The 8051 serial port can be configured to detect bytes with the address bit set, automatically ignoring all others. Pins INT0 and INT1 are interrupts configured respectively as high-priority, falling-edge triggered and low-priority, low-level triggered. The remaining 12 I/O pins output TTL-level control signals to 12 actuators.

There are several ways to implement the sensor matrix circuitry, all logically similar. Figure 20a shows one possibility. Each of the 64 sensors consists of a pair of simple switch contacts in series with a diode to permit multiple contact closures throughout the matrix.

The scan lines from Port 1 provide eight un-encoded active-high scan signals for enabling columns of the matrix. The return lines on rows where a contact is closed are pulled high and read as logic ones. Open return lines are pulled to ground by one of the 40 kΩ resistors and are read as zeroes. (The resistor values must be chosen to ensure all return lines are pulled above the 2.0V logic threshold, even in the worst-case,

where all contacts in an enabled column are closed.) Since P0 is provided open-collector outputs and high-impedance MOS inputs its input loading may be considered negligible.

The circuits in Figures 20b–20d are variations on this theme. When input signals must be electrically isolated from the computer circuitry as in noisy industrial environments, phototransistors can replace the switch diode pairs and provide optical isolation as in Figure 20b. Additional opto-isolators could also be used on the control output and special signal lines.

The other circuits assume that input signals are already at TTL levels. Figure 20c uses octal three-state buffers enabled by active-low scan signals to gate eight signals onto Port 0. Port 0 is available for memory expansion or peripheral chip interfacing between sensor matrix scans. Eight-to-one multiplexers in Figure 20d select one of eight inputs for each return line as determined by encoded address bits output on three pins of Port 1. (Five more output pins are thus freed for more control functions.) Each output can drive at least one standard TTL or up to 10 low-power TTL loads without additional buffering.

Going back to the original matrix circuit, Figure 21 shows the method used to scan the sensor matrix. Two complete bit maps are maintained in the bit-addressable region of the RAM: one for the current state and one for the previous state read for each sensor. If the need arises, the program could then sense input transitions and or debounce contact closures by comparing each bit with its earlier value.

The code in Example 3 implements the scanning algorithm for the circuits in Figure 20a. Each column is enabled by setting a single bit in a field of zeroes. The bit maps are positive logic: ones represent contacts that are closed or isolators turned on.

Example 3.

```

INPUT_SCAN:      ;SUBROUTINE TO READ
                  ;CURRENT STATE
                  ;OF 64 SENSORS AND
                  ;SAVE IN RAM 20H-27H
MOV R0,#20H      ;INITIALIZE
                  ;POINTERS
MOV R1,#28H      ;FOR BIT MAP
MOV A,#80H       ;BASES
                  ;SET FIRST BIT
                  ;IN ACC
SCAN: MOV P1,A    ;OUTPUT TO SCAN
RR A             ;LINES
                ;SHIFT TO ENABLE
                ;NEXT COLUMN
                ;NEXT
MOV R2,A         ;REMEMBER CUR-
                ;RENT SCAN
                ;POSITION
MOV A,P0         ;READ RETURN
XCH A,@R0        ;LINES
                ;SWITCH WITH
                ;PREVIOUS MAP
                ;BITS
MOV @R1,A        ;SAVE PREVIOUS
                ;STATE AS WELL
                ;BUMP POINTERS
INC R0
INC R1
MOV A,R2         ;RELOAD SCAN
                ;LINE MASK
JNB ACC,7;SCAN;LOOP UNTIL ALL
                ;EIGHT COLUMNS
                ;READ
RET

```

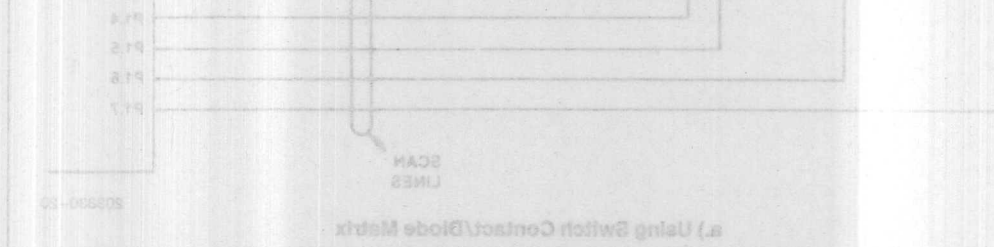


Figure 20. Sensor Matrix Implementation Methods

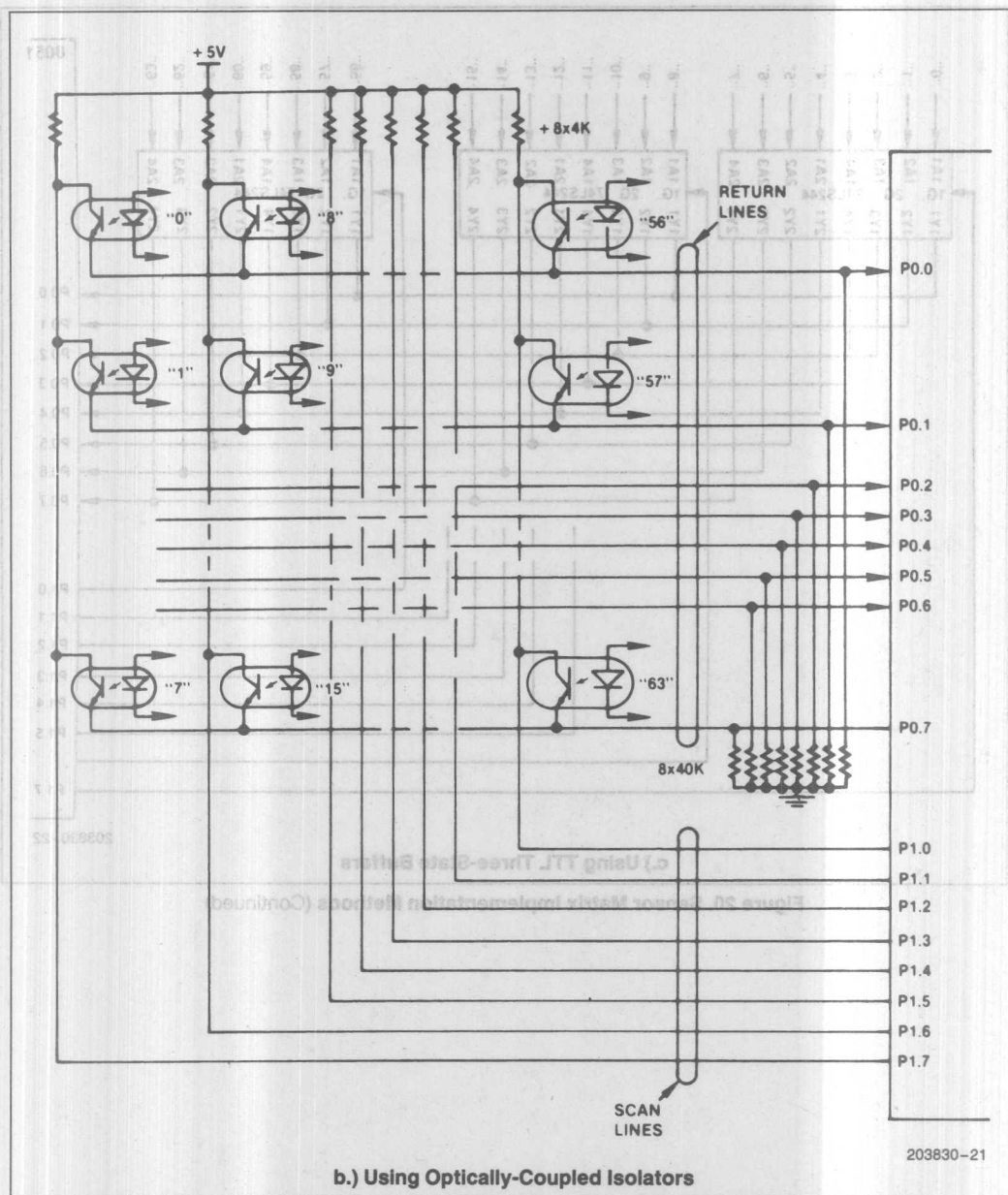


Figure 20. Sensor Matrix Implementation Methods (Continued)

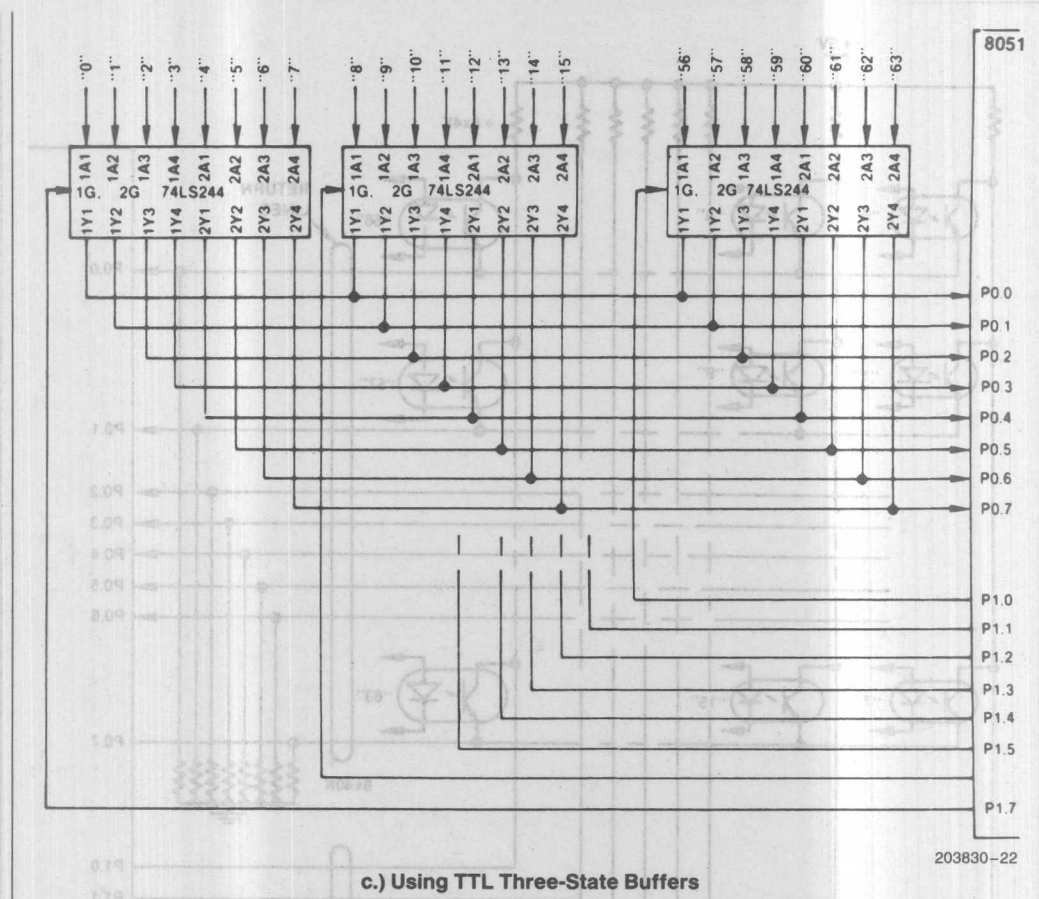


Figure 20. Sensor Matrix Implementation Methods (Continued)

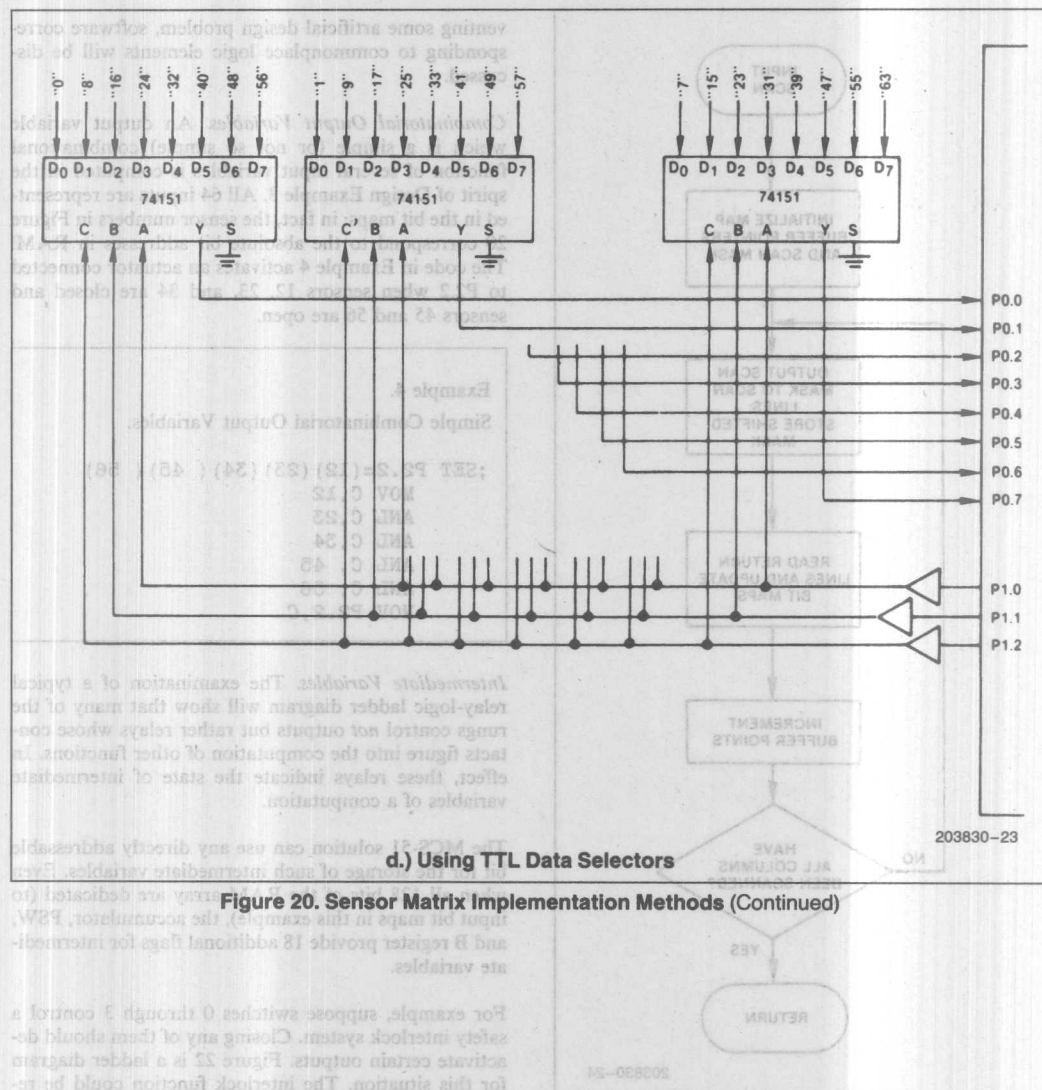


Figure 20. Sensor Matrix Implementation Methods (Continued)

What happens after the sensors have been scanned depends on the individual application. Rather than in-

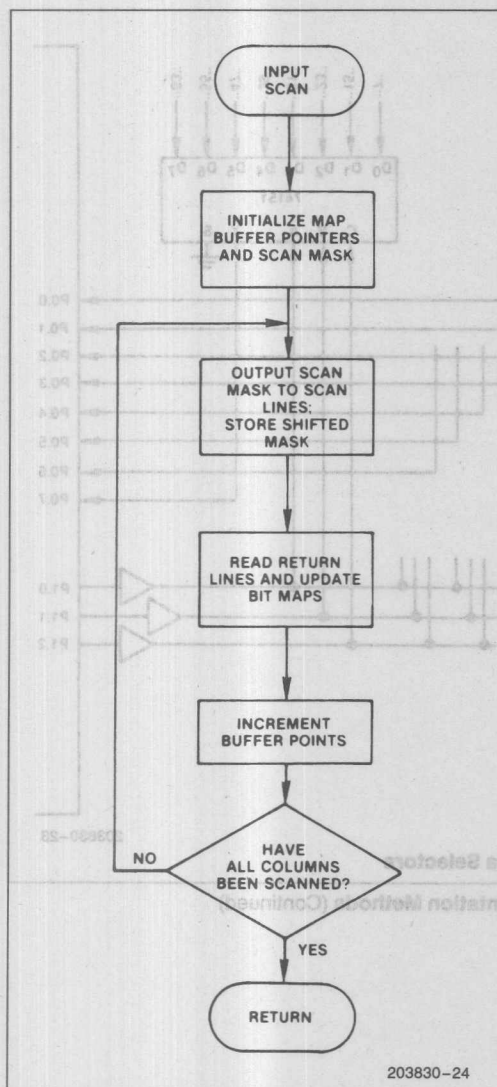


Figure 21. Flowchart for Reading in Sensor Matrix

What happens after the sensors have been scanned depends on the individual application. Rather than in-

venting some artificial design problem, software corresponding to commonplace logic elements will be discussed.

Combinatorial Output Variables. An output variable which is a simple (or not so simple) combinational function of several input variables is computed in the spirit of Design Example 3. All 64 inputs are represented in the bit maps: in fact, the sensor numbers in Figure 20 correspond to the absolute bit addresses in RAM! The code in Example 4 activates an actuator connected to P2.2 when sensors 12, 23, and 34 are closed and sensors 45 and 56 are open.

Example 4.

Simple Combinatorial Output Variables.

```

;SET P2.2=(12)(23)(34)(45)(56)
MOV C,12
ANL C,23
ANL C,34
ANL C,45
ANL C,56
MOV P2.2,C
  
```

Intermediate Variables. The examination of a typical relay-logic ladder diagram will show that many of the rungs control *not* outputs but rather relays whose contacts figure into the computation of other functions. In effect, these relays indicate the state of intermediate variables of a computation.

The MCS-51 solution can use any directly addressable bit for the storage of such intermediate variables. Even when all 128 bits of the RAM array are dedicated (to input bit maps in this example), the accumulator, PSW, and B register provide 18 additional flags for intermediate variables.

For example, suppose switches 0 through 3 control a safety interlock system. Closing any of them should deactivate certain outputs. Figure 22 is a ladder diagram for this situation. The interlock function could be recomputed for every output affected, or it may be computed once and save (as implied by the diagram). As the program proceeds this bit can qualify each output.

Example 5. Incorporating Override signal into actuator outputs.

```

CALL INPUT_SCAN
MOV C,0
ORL C,1
ORL C,2
ORL C,3
MOV FO,C
.....
; COMPUTE FUNCTION 0
;
ANL C, FO
MOV PLO,C
.....
; COMPUTE FUNCTION 1
;
ANL C, FO
MOV P1.1,C
.....
; COMPUTE FUNCTION 2
;
ANL C, FO
MOV P1.2,C
.....

```

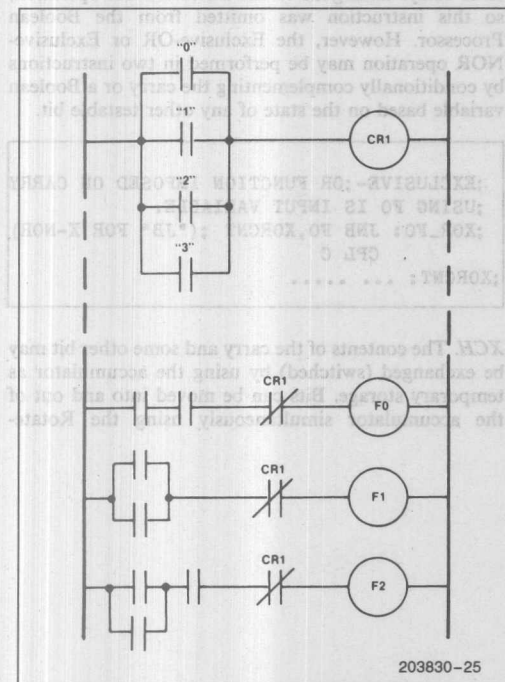


Figure 22. Ladder Diagram for Output Override Circuitry

Latching Relays. A latching relay can be forced into either the ON or OFF state by two corresponding input signals, where it will remain until forced onto the opposite state—analogue to a TTL Set/Reset flip-flop. The relay is used as an intermediate variable for other calculations. In the previous example, the emergency condition could be remembered and remain active until an “emergency cleared” button is pressed.

Any flag or addressable bit may represent a latching relay with a few lines of code (see Example 6).

Example 6. Simulating a latching relay.

```

;I_SET SET FLAG 0 IF C=1
I_SET: ORL C,FO
      MOV FO,C
;
;I_RSET RESET FLAG 0 IF C=1
I_RSET: CPS C
      ANL C,FO
      MOV FO,C

```

Time Delay Relays. A time delay relay does not respond to an input signal until it has been present (or absent) for some predefined time. For example, a ballast or load resistor may be switched in series with a D.C. motor when it is first turned on, and shunted from the circuit after one second. This sort of time delay may be simulated by an interrupt routine driven by one of the two 8051 timer counters. The procedure followed by the routine depends heavily on the details of the exact function needed: time-outs or time delays with resettable or non-resettable inputs are possible. If the interrupt routine is executed every 10 milliseconds the code in Example 7 will clear an intermediate variable set by the background program after it has been active for two seconds.

Example 7. Code to clear USRFLG after a fixed time delay.

```

JNB USR_FLG,NXTTST
DJNZ DLAY_COUNT,NXTTST
CLR USR_FLG
MOV DLAY_COUNT,#200
NXTTST; ;...

```

Serial Interface to Remote Processor. When it detects emergency conditions represented by certain input combinations (such as the earlier Emergency Override), the controller could shut down the machine immediately and/or alert the host processor via the serial port. Code bytes indicating the nature of the problem could be transmitted to a central computer. In fact, at 17,000 bytes-per-second, the entire contents of both bit maps could be sent to the host processor for further analysis in less than a millisecond! If the host decides that conditions warrant, it could alert other remote processors in the system that a problem exists and specify which shut-down sequence each should initiate. For more information on using the serial port, consult the MCS-51 User's Manual.

Response Timing

One difference between relay and programmed industrial controllers (when each is considered as a "black box") is their respective reaction times to input changes. As reflected by a ladder diagram, relay systems contain a large number of "rungs" operating in parallel. A change in input conditions will begin propagating through the system immediately, possibly affecting the output state within milliseconds.

Software, on the other hand, operates sequentially. A change in input states will not be detected until the next time an input scan is performed, and will not affect the outputs until that section of the program is reached. For that reason the raw speed of computing the logical functions is of extreme importance.

Here the Boolean processor pays off. *Every instruction mentioned in this Note* completes in one or two microseconds—the *minimum* instruction execution time for many other microcontrollers! A ladder diagram containing a hundred rungs, with an average of four contacts per rung can be replaced by approximately five hundred lines of software. A complete pass through the entire matrix scanning routine and all computations would require about a millisecond: less than the time it takes for most relays to change state.

Example 7. Code to clear USERLQ after a fixed time delay.

```

M0TST:  ...
MOV  DLYA_COUNT,#200
CLR  USER_LQ
DINE DLYA_COUNT,M0TST
JNB  USER_LQ,M0TST

```

A programmed controller which simulates each Boolean function with a subroutine would be less efficient by at least an order of magnitude. Extra software is needed for the simulation routines, and each step takes longer to execute for three reasons: several byte-wide logical instructions are executed per user program step (rather than one Boolean operation); most of those instructions take longer to execute with microprocessors performing multiple off-chip accesses; and calling and returning from the various subroutines requires overhead for stack operations.

In fact, the speed of the Boolean Processor solution is likely to be much faster than the system requires. The CPU might use the time left over to compute feedback parameters, collect and analyze execution statistics, perform system diagnostics, and so forth.

Additional Functions and Uses

With the building-block basics mentioned above many more operations may be synthesized by short instruction sequences.

Exclusive-OR. There are no common mechanical devices or relays analogous to the Exclusive-OR operation, so this instruction was omitted from the Boolean Processor. However, the Exclusive-OR or Exclusive-NOR operation may be performed in two instructions by conditionally complementing the carry or a Boolean variable based on the state of any other testable bit.

```

;EXCLUSIVE-OR FUNCTION IMPOSED ON CARRY
;USING FO IS INPUT VARIABLE.
;XOR_FO: JNB FO,XORCNT ;("JB" FOR X-NOR)
        CPL C
;XORCNT: ... ..

```

XCH. The contents of the carry and some other bit may be exchanged (switched) by using the accumulator as temporary storage. Bits can be moved into and out of the accumulator simultaneously using the Rotate-

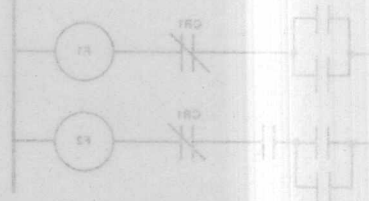


Figure 25. Ladder Diagram for Output Override Circuitry

through-carry instructions, though this would alter the accumulator data.

```

;EXCHANGE CARRY WITH USRFLG
XCHBIT: RLC    A
        MOV     C,USR_FLG
        RRC     A
        MOV     USR_FLG,C
        RLC     A

```

Extended Bit Addressing. The 8051 can directly address 144 general-purpose bits for all instructions in Figure 3b. Similar operations may be extended to any bit anywhere on the chip with some loss of efficiency.

The logical operations AND, OR, and Exclusive-OR are performed on byte variables using six different addressing modes, one of which lets the source be an immediate mask, and the destination any directly addressable byte. Any bit may thus be set, cleared, or complemented with a three-byte, two-cycle instruction if the mask has all bits but one set or cleared.

Byte variables, registers, and indirectly addressed RAM may be moved to a bit addressable register (usually the accumulator) in one instruction. Once transferred, the bits may be tested with a conditional jump, allowing any bit to be polled in 3 microseconds—still much faster than most architectures—or used for logical calculations. (This technique can also simulate additional bit addressing modes with byte operations.)

Parity of bytes or bits. The parity of the current accumulator contents is always available in the PSW, from whence it may be moved to the carry and further processed. Error-correcting Hamming codes and similar applications require computing parity on groups of isolated bits. This can be done by conditionally complementing the carry flag based on those bits or by gathering the bits into the accumulator (as shown in the DES example) and then testing the parallel parity flag.

Multiple byte shift and CRC codes

Though the 8051 serial port can accommodate eight- or nine-bit data transmissions, some protocols involve much longer bit streams. The algorithms presented in

Design Example 2 can be extended quite readily to 16 or more bits by using multi-byte input and output buffers.

Many mass data storage peripherals and serial communications protocols include Cyclic Redundancy (CRC) codes to verify data integrity. The function is generally computed serially by hardware using shift registers and Exclusive-OR gates, but it can be done with software. As each bit is received into the carry, appropriate bits in the multi-byte data buffer are conditionally complemented based on the incoming data bit. When finished, the CRC register contents may be checked for zero by OR'ing the two bytes in the accumulator.

4.0 SUMMARY

A truly unique facet of the Intel MCS-51 microcomputer family design is the collection of features optimized for the one-bit operations so often desired in real-world, real-time control applications. Included are 17 special instructions, a Boolean accumulator, implicit and direct addressing modes, program and mass data storage, and many I/O options. These are the world's first single-chip microcomputers able to efficiently manipulate, operate on, and transfer either bytes or individual bits as data.

This Application Note has detailed the information needed by a microcomputer system designer to make full use of these capabilities. Five design examples were used to contrast the solutions allowed by the 8051 and those required by previous architectures. Depending on the individual application, the 8051 solution will be easier to design, more reliable to implement, debug, and verify, use less program memory, and run up to an order of magnitude faster than the same function implemented on previous digital computer architectures.

Combining byte- and bit-handling capabilities in a single microcomputer has a strong synergistic effect: the power of the result exceeds the power of byte- and bit-processors laboring individually. Virtually all user applications will benefit in some way from this duality. Data intensive applications will use bit addressing for test pin monitoring or program control flags; control applications will use byte manipulation for parallel I/O expansion or arithmetic calculations.

It is hoped that these design examples give the reader an appreciation of these unique features and suggest ways to exploit them in his or her own application.

APPENDIX A

Automobile Turn-Indicator Controller Program Listing

ISIS-II MCS-51 MACRO ASSEMBLER V1.0
OBJECT MODULE PLACED IN FO.AP70.HEX
ASSEMBLER INVOKED BY: f1.asm51 ap70 src date(328)

LOC	OBJ	LINE	SOURCE
		1	\$XREF TITLE(AP-70 APPENDIX)
		2	*****
		3	
		4	THE FOLLOWING PROGRAM USES THE BOOLEAN INSTRUCTION SET
		5	OF THE INTEL 8051 MICROCOMPUTER TO PERFORM A NUMBER OF
		6	AUTOMOTIVE DASHBOARD CONTROL FUNCTIONS RELATING TO
		7	TURN SIGNAL CONTROL, EMERGENCY BLINKERS, BRAKE LIGHT
		8	CONTROL, AND PARKING LIGHT OPERATION.
		9	THE ALGORITHMS AND HARDWARE ARE DESCRIBED IN DESIGN
		10	EXAMPLE #4 OF INTEL APPLICATION NOTE AP-70.
		11	"USING THE INTEL MCS-51(TM)
		12	BOOLEAN PROCESSING CAPABILITIES"
		13	*****
		14	
		15	
		16	INPUT PIN DECLARATIONS:
		17	(ALL INPUTS ARE POSITIVE-TRUE LOGIC
		18	INPUTS ARE HIGH WHEN RESPECTIVE SWITCH CONTACT IS CLOSED)
		19	
0090		20	BRAKE BIT P1.0 ; BRAKE PEDAL DEPRESSED
0091		21	EMERG BIT P1.1 ; EMERGENCY BLINKER ACTIVATED
0092		22	PARK BIT P1.2 ; PARKING LIGHTS ON
0093		23	L_TURN BIT P1.3 ; TURN LEVER DOWN
0094		24	R_TURN BIT P1.4 ; TURN LEVER UP
		25	
		26	OUTPUT PIN DECLARATIONS:
		27	(ALL OUTPUTS ARE POSITIVE TRUE LOGIC
		28	BULB IS TURNED ON WHEN OUTPUT PIN IS HIGH.)
		29	
0095		30	L_FRNT BIT P1.5 ; FRONT LEFT-TURN INDICATOR
0096		31	R_FRNT BIT P1.6 ; FRONT RIGHT-TURN INDICATOR
0097		32	L_DASH BIT P1.7 ; DASHBOARD LEFT-TURN INDICATOR
00A0		33	R_DASH BIT P2.0 ; DASHBOARD RIGHT-TURN INDICATOR
00A1		34	L_REAR BIT P2.1 ; REAR LEFT-TURN INDICATOR
00A2		35	R_REAR BIT P2.2 ; REAR RIGHT-TURN INDICATOR
		36	
00A3		37	S_FAIL BIT P2.3 ; ELECTRICAL SYSTEM FAULT INDICATOR
		38	
		39	INTERNAL VARIABLE DEFINITIONS:
		40	
0020		41	SUB_DIV DATA 20H ; INTERRUPT RATE SUBDIVIDER
0000		42	HI_FREQ BIT SUB_DIV.0 ; HIGH-FREQUENCY OSCILLATOR BIT
0007		43	LO_FREQ BIT SUB_DIV.7 ; LOW-FREQUENCY OSCILLATOR BIT
		44	
00D1		45	DIM BIT PSW.1 ; PARKING LIGHTS ON FLAG
		46	
		47	=====
		48	+1 \$EJECT

LOC	OBJ	LINE	SOURCE	END	
0000	020040	49	ORG	0000H	RESET VECTOR ON INTERRUPT ROUTINE
		50	LUMP	INIT	BECAUSE 68H
		51			
0008	758CF0	52	ORG	000BH	TIMER 0 SERVICE VECTOR
000E	C0D0	53	MOV	TH0, #-16	HIGH TIMER BYTE ADJUSTED TO CONTROL INT. RATE
0010	0154	54	PUSH	PSW	EXECUTE CODE TO SAVE ANY REGISTERS USED BELOW
		55	AJMP	UPDATE	AND (CONTINUE WITH REST OF ROUTINE)
		56			
0040	758A00	57	ORG	0040H	INITIAL LOW ORDER
0043	758CF0	58	INIT: MOV	TLO, #0	ZERO LOADED INTO LOW-ORDER BYTE AND
0046	758961	59	MOV	TH0, #-16	-16 IN HIGH-ORDER BYTE GIVES 4 MSEC PERIOD
		60	MOV	TMD, #01100001B	8-BIT AUTO RELOAD COUNTER MODE FOR TIMER 1,
		61	MOV	MDR, #0	16-BIT TIMER MODE FOR TIMER 0 SELECTED
0049	7520F4	62	MOV	SUB_DIV, #244	SUBDIVIDE INTERRUPT RATE BY 244 FOR 1 HZ
004C	D2A9	63	SETB	ETO	USE TIMER 0 OVERFLOWS TO INTERRUPT PROGRAM
004E	D2AF	64	SETB	EA	CONFIGURE IE TO GLOBALLY ENABLE INTERRUPTS
0050	D28C	65	SETB	TRO	KEEP INSTRUCTION CYCLE COUNT UNTIL OVERFLOW
0052	80FE	66	SJMP	\$	START BACKGROUND PROGRAM EXECUTION
		67			
		68			
0054	D5203B	69	UPDATE: DJNZ	SUB_DIV, TOSERV	EXECUTE SYSTEM TEST ONLY ONCE PER SECOND
0057	7520F4	70	MOV	SUB_DIV, #244	GET VALUE FOR NEXT ONE SECOND DELAY AND
		71			GO THROUGH ELECTRICAL SYSTEM TEST CODE:
005A	4390E0	72	ORL	P1, #11100000B	SET CONTROL OUTPUTS HIGH
005D	43A007	73	ORL	P2, #00000111B	SWITCH DRIVE LEDS ON
0060	C295	74	CLR	L_FRNT	FLOAT DRIVE COLLECTOR
0062	20842B	75	JB	TO, FAULT	TO SHOULD BE PULLED LOW
0065	D295	76	SETB	L_FRNT	PULL COLLECTOR BACK DOWN
0067	C297	77	CLR	L_DASH	REPEAT SEQUENCE FOR L_DASH,
0069	208421	78	JB	TO, FAULT	ADD IN SWIRLING LIGHT FUNCTION
006C	D297	79	SETB	L_DASH	SWIRLING LIGHT ON
006E	C2A1	80	CLR	L_REAR	L_REAR,
0070	20841A	81	JB	TO, FAULT	TO SHOULD BE PULLED LOW
0073	D2A1	82	SETB	L_REAR	PULL COLLECTOR BACK DOWN
0075	C296	83	CLR	R_FRNT	R_FRNT,
0077	208413	84	JB	TO, FAULT	TO SHOULD BE PULLED LOW
007A	D296	85	SETB	R_FRNT	REPEAT SEQUENCE FOR R_FRNT,
007C	C2A0	86	CLR	R_DASH	R_DASH,
007E	20840C	87	JB	TO, FAULT	TO SHOULD BE PULLED LOW
0081	D2A0	88	SETB	R_DASH	PULL COLLECTOR BACK DOWN
0083	C2A2	89	CLR	R_REAR	R_REAR,
0085	208405	90	JB	TO, FAULT	TO SHOULD BE PULLED LOW
0088	D2A2	91	SETB	R_REAR	REPEAT SEQUENCE FOR R_REAR,
		92			
008A	208402	93			WITH ALL COLLECTORS GROUNDED, TO SHOULD BE HIGH
008D	B2A3	94			IF SO, CONTINUE WITH INTERRUPT ROUTINE.
		95			
008A	208402	96	JB	TO, TOSERV	TO SHOULD BE PULLED LOW
008D	B2A3	97	FAULT: CPL	S_FAIL	ELECTRICAL FAILURE PROCESSING ROUTINE
		98			(TOGGLE INDICATOR ONCE PER SECOND)
		99	+1	\$EJECT	

LOC	OBJ	LINE	SOURCE
008D 85W3		100	CONTINUE WITH INTERRUPT PROCESSING.
008E 508+05		101	
		102	1) COMPUTE LOW BULB INTENSITY WHEN PARKING LIGHTS ARE ON.
		103	
008F A201		104	TOSERV: MOV C, SUB_DIV 1 ; START WITH 50 PERCENT.
0091 8200		105	ANL C, SUB_DIV 0 ; MASK DOWN TO 25 PERCENT.
0093 7202		106	ORL C, SUB_DIV 2 ; BUILD BACK TO 62.5 PERCENT.
0095 8292		107	ANL C, PARK ; GATE WITH PARKING LIGHT SWITCH.
0097 92D1		108	MOV DIM, C ; AND SAVE IN TEMP. VARIABLE.
		109	
0098 85W3		110	2) COMPUTE AND OUTPUT LEFT-HAND DASHBOARD INDICATOR.
009E 508+0C		111	
0099 A293		112	MOV C, L_TURN ; SET CARRY IF TURN
009B 7291		113	ORL C, EMERG ; OR EMERGENCY SELECTED.
009D 8207		114	ANL C, LO_FREQ ; IF SO, GATE IN 1 HZ SIGNAL
009F 9297		115	MOV L_DASH, C ; AND OUTPUT TO DASHBOARD.
		116	
00A0 508+1V		117	3) COMPUTE AND OUTPUT LEFT-HAND FRONT TURN SIGNAL.
009E C5V1		118	
00A1 92D5		119	MOV FO, C ; SAVE FUNCTION SO FAR.
00A3 72D1		120	ORL C, DIM ; ADD IN PARKING LIGHT FUNCTION
00A5 9295		121	MOV L_FRNT, C ; AND OUTPUT TO TURN SIGNAL.
		122	
0095 508+5B		123	4) COMPUTE AND OUTPUT LEFT-HAND REAR TURN SIGNAL.
009C C5A2		124	
00A7 A290		125	MOV C, BRAKE ; GATE BRAKE PEDAL SWITCH
00A9 8093		126	ANL C, /L_TURN ; WITH TURN LEVER.
00AB 72D5		127	ORL C, FO ; INCLUDE TEMP. VARIABLE FROM DASH
00AD 72D1		128	ORL C, DIM ; AND PARKING LIGHT FUNCTION
00AF 92A1		129	MOV L_REAR, C ; AND OUTPUT TO TURN SIGNAL.
		130	
		131	5) REPEAT ALL OF ABOVE FOR RIGHT-HAND COUNTERPARTS.
00B5 80+E		132	
00B1 A294		133	MOV C, R_TURN ; SET CARRY IF TURN
00B3 7291		134	ORL C, EMERG ; OR EMERGENCY SELECTED.
00B5 8207		135	ANL C, LO_FREQ ; IF SO, GATE IN 1 HZ SIGNAL
00B7 92A0		136	MOV R_DASH, C ; AND OUTPUT TO DASHBOARD.
00B9 92D5		137	MOV FO, C ; SAVE FUNCTION SO FAR.
00BB 72D1		138	ORL C, DIM ; ADD IN PARKING LIGHT FUNCTION
00BD 9296		139	MOV L_FRNT, C ; AND OUTPUT TO TURN SIGNAL.
00BF A290		140	MOV C, BRAKE ; GATE BRAKE PEDAL SWITCH
00C1 8094		141	ANL C, /R_TURN ; WITH TURN LEVER.
00C3 72D5		142	ORL C, FO ; INCLUDE TEMP. VARIABLE FROM DASH
00C5 72D1		143	ORL C, DIM ; AND PARKING LIGHT FUNCTION
00C7 92A2		144	MOV R_REAR, C ; AND OUTPUT TO TURN SIGNAL.
		145	
00C8		146	RESTORE STATUS REGISTER AND RETURN.
		147	
00C9 D0D0		148	POP PSW ; RESTORE PSW
00CB 32		149	RETI ; AND RETURN FROM INTERRUPT ROUTINE
		150	
END		151	END

XREF SYMBOL TABLE LISTING

NAME	TYPE	VALUE AND REFERENCES
BRAKE	N BSEG	0090H 20# 125 140
DIM	N BSEG	00D1H 45# 108 120 128 138 143
EA	N BSEG	00AFH 64
EMERG	N BSEG	0091H 21# 113 134
ETO	N BSEG	00A9H 63
F0	N BSEG	00D5H 119 127 137 142
FAULT	L CSEG	008DH 75 78 81 84 87 90 97#
HI_FREQ	N BSEG	0000H 42#
INIT	L CSEG	0040H 50 58#
L_DASH	N BSEG	0097H 32# 77 79 115
L_FRNT	N BSEG	0095H 30# 74 76 121
L_REAR	N BSEG	00A1H 34# 80 82 129
L_TURN	N BSEG	0093H 23# 112 126
LO_FREQ	N BSEG	0007H 43# 114 135
P1	N DSEG	0090H 20 21 22 23 24 30 31 32 72
P2	N DSEG	00A0H 33 34 35 37 73
PARK	N BSEG	0092H 22# 107
PSW	N DSEG	00D0H 45 54 148
R_DASH	N BSEG	00A0H 33# 86 88 136
R_FRNT	N BSEG	0096H 31# 83 85 139
R_REAR	N BSEG	00A2H 35# 89 91 144
R_TURN	N BSEG	0094H 24# 133 141
S_FAIL	N BSEG	00A3H 37# 97
SUB_DIV	N DSEG	0020H 41# 42 43 62 69 70 104 105 106
T0	N BSEG	00B4H 75 78 81 84 87 90 96
TOSERV	L CSEG	00BFH 69 96 104#
TH0	N DSEG	00BCH 53 59
TLO	N DSEG	00BAH 58
TMOD	N DSEG	00B9H 60
TRO	N BSEG	00BCH 65
UPDATE	L CSEG	0054H 55 69#

ASSEMBLY COMPLETE. NO ERRORS FOUND

203830-29



APPLICATION NOTE

AP-223

8051 Based CRT Terminal Controller

Michael A. Shafer
Microcontroller Applications

October 1984

1.0 INTRODUCTION

This is the third application note that Intel has produced on CRT terminal controllers. The first Ap Note (ref. 1), written in 1977, used the 8080 as the CPU and required 41 packages including 11 LSI devices. In 1979, another application note (ref. 2) using the 8085 as the controller was produced and the chip count decreased to 20 with 11 LSI devices.

Advancing technology has integrated a complete system onto a single device that contains a CPU, program memory, data memory, serial communication, interrupt controller, and I/O. These "computer-on-a-chip" devices are known as microcontrollers. Intel's MCS[®]-51 microcontroller was chosen for this application because of its highly integrated functions. This CRT terminal design uses 12 packages with only 4 LSI devices.

This application note has been divided into five general sections:

- 1) CRT Terminal Basics
- 2) 8051 Description
- 3) 8276 Description
- 4) Design Background
- 5) System Description

2.0 CRT TERMINAL BASICS

A terminal provides a means for humans to communicate with a computer. Terminals may be as simple as a LED display and a couple of push buttons, or it may be an elaborate graphics system that contains a full function keyboard with user programmable keys, color CRT and several processors controlling its functions. This application note describes a basic low cost terminal containing a black and white CRT display, full function keyboard and a serial interface.

2.1 CRT Description

A raster scan CRT displays its images by generating a series of lines (raster) across the face of the tube. The electron beam usually starts at the top left hand corner moves left to right, back to the left of the screen, moves down one row and continues on to the right. This is repeated until the lower right hand of the screen is reached. Then the beam returns to the top left hand corner and refreshes the screen. The beam forms a zigzag pattern as shown in Figure 2.1.0.

Two independent operating circuits control this movement across the screen. The horizontal oscillator controls the left to right motion of the beam while the vertical controls the top to bottom movement. The vertical oscillator also tells the beam when to return to the upper left hand corner or "home" position.

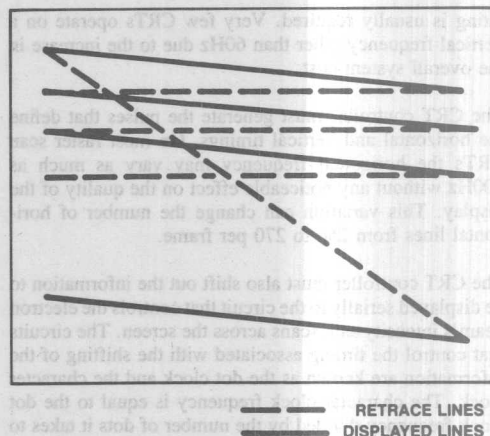


Figure 2.1.0 Raster Scan

As the electron beam moves across the screen under the control of the horizontal oscillator, a third circuit controls the current entering the electron gun. By varying the current, the image may be made as bright or as dim as the user desires. This control is also used to turn the beam off or "blank the screen".

When the beam reaches the right hand side of the screen, the beam is blanked so it does not appear on the screen as it returns to the left side. This "retrace" of the beam is at a much faster rate than it traveled across the screen to generate the image.

The time it takes to scan the whole screen and return to the home position is referred to as a "frame". In the United States, commercial television broadcast uses a horizontal sweep frequency of 15,750Hz which calculates out to 63.5 microseconds per line. The frame time is equal to 16.67 milliseconds or 60Hz vertical sweep frequency.

Although this is the commercial standard, many CRT displays operate from 18KHz to 30KHz horizontal frequency. As the horizontal frequency increases, the number of lines per frame increases. This increase in lines or resolution is needed for graphic displays and on special text editors that display many more lines of text than the standard 24 or 25 character lines.

Since the United States operates on a 60Hz A.C. power line frequency, most CRT monitors use 60Hz as the vertical frequency. The use of 60Hz as the vertical frequency allows the magnetic and electrical variations that can modulate the electron beam to be synchronized with the display, thus they go unnoticed. If a frequency other than 60Hz is used, special shielding and power supply regu-

lating is usually required. Very few CRTs operate on a vertical frequency other than 60Hz due to the increase in the overall system cost.

The CRT controller must generate the pulses that define the horizontal and vertical timings. On most raster scan CRTs the horizontal frequency may vary as much as 500Hz without any noticeable effect on the quality of the display. This variation can change the number of horizontal lines from 256 to 270 per frame.

The CRT controller must also shift out the information to be displayed serially to the circuit that controls the electron beam's intensity as it scans across the screen. The circuits that control the timing associated with the shifting of the information are known as the dot clock and the character clock. The character clock frequency is equal to the dot clock frequency divided by the number of dots it takes to form a character in the horizontal axis. The dot clock frequency is calculated by the following equation:

$$\text{Dot Clock (Hz)} = (N + R) * D * L * F$$

where

N is the number of displayed characters per row,
R is the number of character times for the retrace,
D is the number of dots per character in the horizontal axis,
L is the number of horizontal lines per frame,
F is the frame rate in Hz.

In this design N = 80, R = 20, D = 7, L = 270, and F = 60Hz. Plugging in the numbers results in a dot clock frequency of 11.34MHz.

The retrace number may vary on each design because it is used to set the left and right hand margins on the CRT. The number of dots per character is chosen by the designer to meet the system needs. In this design, a 5 x 7 dot matrix and 2 blank dots between each character (see Figure 2.1.1) makes D equal to 5 + 2 = 7.

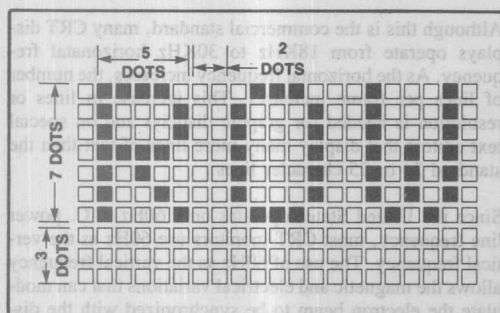


Figure 2.1.1 5 x 7 Dot Matrix

The following equation can be used to figure the number of lines per frame:

$$L = (H * Z) + V$$

where

H is the number of horizontal lines per character,
Z is the number of character lines per frame,
V is the number of horizontal line times during the vertical retrace

In this design H is equal to the 7 horizontal dots per character plus 3 blank dots between each row which adds up to 10. Also 25 lines of characters are displayed, so Z = 25. The vertical retrace time is variable to set the top and bottom margins on the CRT and in this design is equal to 20. Plugging in the numbers gives L = 270 lines per frame.

2.2 Keyboard

A keyboard is the common way a human enters commands and data to a computer. A keyboard consists of a matrix of switches that are scanned every couple of milliseconds by a keyboard controller to determine if one of the keys has been pressed. Since the keyboard is made up of mechanical switches that tend to bounce or "make and break" contact everytime they are pressed, debouncing of the switches must also be a function of the keyboard controller. There are dedicated keyboard controllers available that do everything from scanning the keyboard, debouncing the keys, decoding the ASCII code for that key closure to flagging the CPU that a valid key has been depressed. The keyboard controller may present the information to the CPU in parallel form or in a serial data stream.

This Application Note integrates the function of the keyboard controller into the 8051 which is also the terminal controller. Provisions have been made to interface the 8051 to a keyboard that uses a dedicated keyboard controller. The 8051 can accept data from the keyboard controller in either parallel or serial format.

2.3 Serial Communications

Communication between a host computer and the CRT terminal can be in either parallel or serial data format. Parallel data transmission is needed in high end graphic terminals where great amounts of information must be transferred.

One can rarely type faster than 120 words per minute, which corresponds to 12 characters per second or 1 character per 83 milliseconds. The utilization of a parallel port cannot justify the cost associated with the drivers and the amount of wire needed to perform this transmission. Full duplex serial data transmission requires 3 wires and two

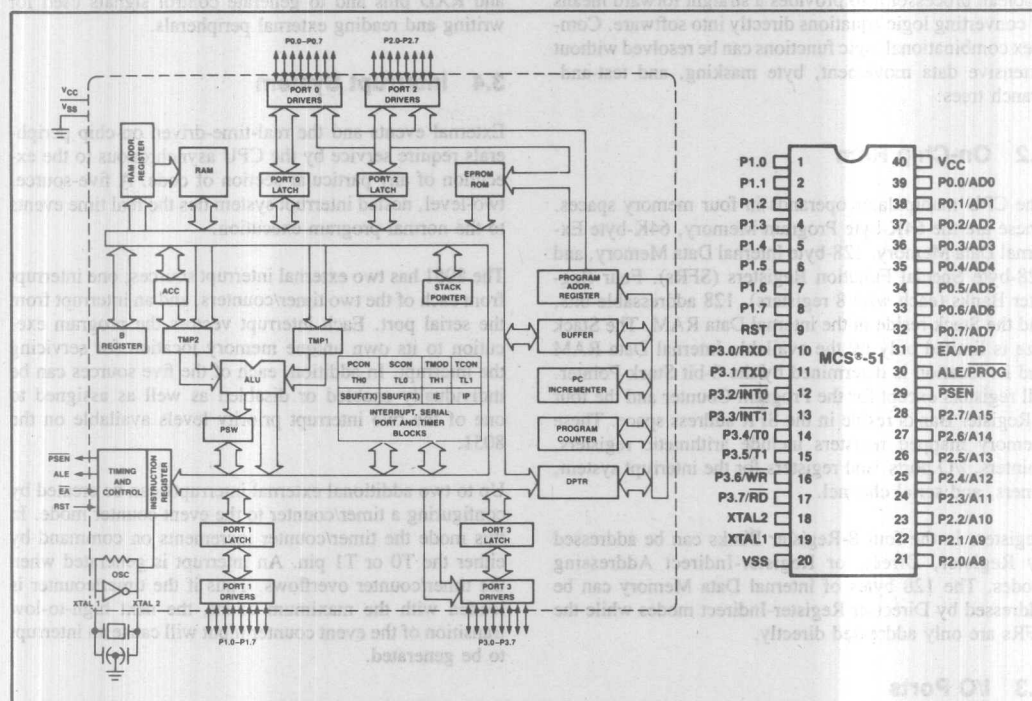


Figure 3.0.0 8051 Block Diagram

drivers to implement the communication channel between the host computer and the terminal. The data rate can be as high as 19200 BAUD in the asynchronous serial format. BAUD rate is the number of bits per second received or transmitted. In the asynchronous serial format, 10 bits of information is required to transmit one character. One character per 500 microseconds or 1,920 characters per second would then be transmitted using 19.2 KBAUD.

This application note uses the 8051 serial port configured for full duplex asynchronous serial data transmission. The software for the 8051 has been written to support variable BAUD rates from 150 BAUD up to 9.6 KBAUD.

3.0 8051 DESCRIPTION

The 8051 is a single chip high-performance microcontroller. A block diagram is shown in figure 3.0.0. The 8051 combines CPU; Boolean processor; 4K \times 8 ROM; 128 \times 8 RAM; 32 I/O lines; two 16-bit timer/ event counters; a five-source, two-priority-level, nested interrupt structure; serial I/O port for either multiprocessor communications, I/O expansion, or full duplex UART; and on-chip oscillator and clock circuits.

3.1 CPU

Efficient use of program memory results from an instruction set consisting of 49 single-byte, 45 two-byte and 17 three-byte instructions. Most arithmetic, logical and branching operations can be performed using an instruction that appends either a short address or a long address. For example, branches may use either an offset that is relative to the program counter which takes two bytes or a direct 16-bit address which takes three bytes to perform. As a result, 64 instructions operate in one machine cycle, 45 in two machine cycles, and the multiply and divide instruction execute in 4 machine cycles.

The 8051 has five addressing modes for source operands: Register, Direct, Register-Indirect, Immediate, and Based-Register-plus Index-Register-Indirect Addressing.

The Boolean Processor can be thought of as a separate one-bit CPU. It has its own accumulator (the carry bit), instruction set for data moves, logic, and control transfer, and its own bit addressable RAM and I/O. The bit-manipulating instructions provide optimum code and speed efficiency for handling on chip peripherals. The

Boolean processor also provides a straight forward means of converting logic equations directly into software. Complex combinational logic functions can be resolved without extensive data movement, byte masking, and test-and-branch trees.

3.2 On-Chip Ram

The CPU manipulates operands in four memory spaces. These are the 64K-byte Program Memory, 64K-byte External Data Memory, 128-byte Internal Data Memory, and 128-byte Special Function Registers (SFRs). Four Register Banks (each with 8 registers), 128 addressable bits, and the Stack reside in the internal Data RAM. The Stack size is limited only by the available Internal Data RAM and its location is determined by the 8-bit Stack Pointer. All registers except for the Program Counter and the four 8-Register Banks reside in the SFR address space. These memory mapped registers include arithmetic registers, pointers, I/O ports, and registers for the interrupt system, timers, and serial channel.

Registers in the four 8-Register Banks can be addressed by Register, Direct, or Register-Indirect Addressing modes. The 128 bytes of internal Data Memory can be addressed by Direct or Register-Indirect modes while the SFRs are only addressed directly.

3.3 I/O Ports

The 8051 has instructions that can treat the 32 I/O lines as 32 individually addressable bits or as 4 parallel 8-bit ports addressable as Ports 0, 1, 2, and 3.

Resetting the 8051 writes a logical 1 to each pin on port 0 which places the output drivers into a high-impedance mode. Writing a logical 0 to a pin forces the pin to ground and sinks current. Re-writing the pin high will place the pin in either an open drain output or high-impedance input mode.

Ports 1, 2, and 3 are known as quasi-bidirectional I/O pins. Resetting the device writes a logical one to each pin. Writing a logical 0 to the pin will force the pin to ground and sink current. Re-writing the pin high will place the pin in an output mode with a weak depletion pullup FET or in the input mode. The weak pullup FET is easily overcome by a TTL output.

Ports 0 and 2 can also be used for off-chip peripheral expansion. Port 0 provides a multiplexed low-order address and data bus while Port 2 contains the high-order address when using external Program Memory or more than 256 byte external Data Memory.

Port 3 pins can also be used to provide external interrupt request inputs, event counter inputs, the serial port TXD

and RXD pins and to generate control signals used for writing and reading external peripherals.

3.4 Interrupt System

External events and the real-time-driven on-chip peripherals require service by the CPU asynchronous to the execution of any particular section of code. A five-source, two-level, nested interrupt system ties the real time events to the normal program execution.

The 8051 has two external interrupt sources, one interrupt from each of the two timer/counters, and an interrupt from the serial port. Each interrupt vectors the program execution to its own unique memory location for servicing the interrupt. In addition, each of the five sources can be individually enabled or disabled as well as assigned to one of the two interrupt priority levels available on the 8051.

Up to two additional external interrupts can be created by configuring a timer/counter to the event counter mode. In this mode the timer/counter increments on command by either the T0 or T1 pin. An interrupt is generated when the timer/counter overflows. Thus if the timer/counter is loaded with the maximum count, the next high-to-low transition of the event counter input will cause an interrupt to be generated.

3.5 Serial Port

The 8051's serial port is useful for linking peripheral devices as well as multiple 8051s through standard asynchronous protocols with full duplex operation. The serial port also has a synchronous mode for expansion of I/O lines using shift registers. This hardware serial port saves ROM code and permits a much higher transmission rate than could be achieved through software. The processor merely needs to read or write the serial buffer in response to an interrupt. The receiver is double buffered to eliminate the possibility of overrun if the processor failed to read the buffer before the beginning of the next frame.

The full duplex asynchronous serial port provides the means of communication with standard UART devices such as CRT terminals and printers.

The reader should refer to the microcontroller handbook for a complete discussion of the 8051 and its various modes of operation.

4.0 8276 DESCRIPTION

The 8276's block diagram and pin configuration are shown in Figure 4.0.0. The following sections describe the general capabilities of the 8276.

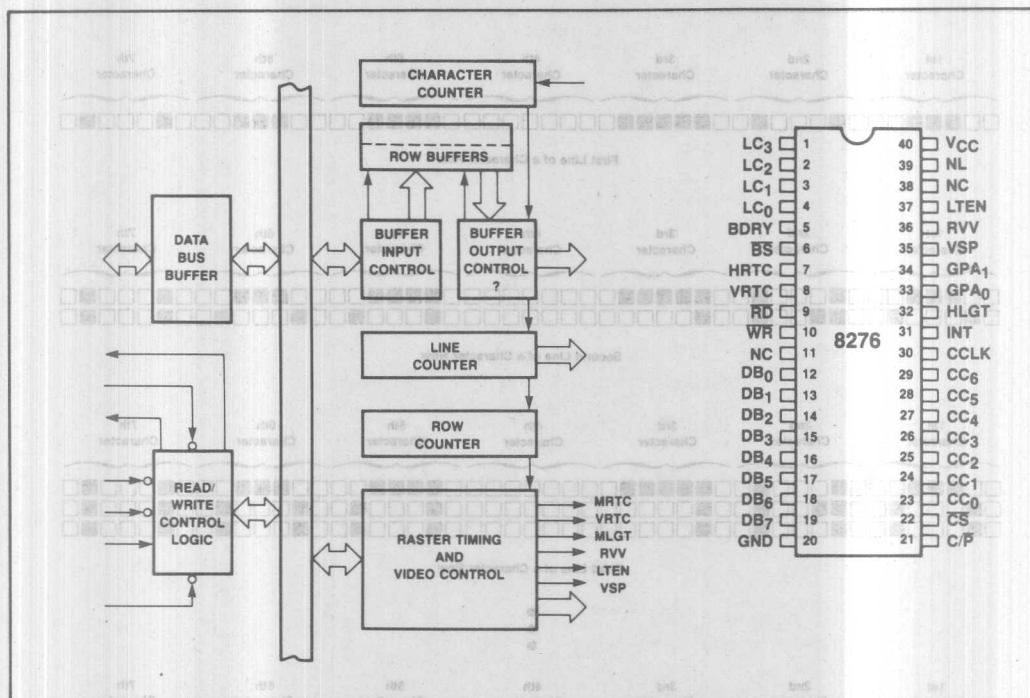


Figure 4.0.0 8276 Block Diagram

4.1 CRT Display Refreshing

The 8276, having been programmed by the system designer for a specific screen format, generates a series of Buffer Ready signals. A row of characters is then transferred by the system controller from the display memory to the 8276's row buffers. The row buffers are filled by deselecting the 8276 \overline{CS} and asserting the BS and WR signals. The 8276 presents the character codes to an external character generator ROM by using outputs CC0-CC6. The parallel data from the outputs of the character generator is converted to serial information that is clocked by external dot timing logic into the video input of the CRT.

The character rows are displayed on the CRT one line at a time. Line count outputs LC0-LC3 select the current line information from the character generator ROM. The display process is illustrated in Figure 4.1.0. This process is repeated for each display character row. At the beginning of the last display row the 8276 generates an interrupt request by raising its INT output line. The interrupt request

is used by the 8051 system controller to reinitialize its load buffer pointers for the next display refresh cycle.

Proper CRT refreshing requires that certain 8276 parameters be programmed at system initialization time. The 8276 has two types of internal registers; the write only Command (CREG) and Parameter (PREG) Registers, and the read only Status Register (SREG). The 8276 expects to receive a command followed by 0 to 4 parameter bytes depending on the command. A summary of the 8276's instruction set is shown in Figure 4.1.1. To access the registers, \overline{CS} must be asserted along with \overline{WR} or \overline{RD} . The status of the C/P pin determines whether the command or parameter registers are selected.

The 8276 allows the designer flexibility in the display format. The display may be from 1 to 80 characters per row, 1 to 64 rows per screen, and 1 to 16 horizontal lines per character row. In addition, four cursor formats are available; blinking, non-blinking, underline, and reverse video. The cursor position is programmable to anywhere on the screen via the Load Cursor command.

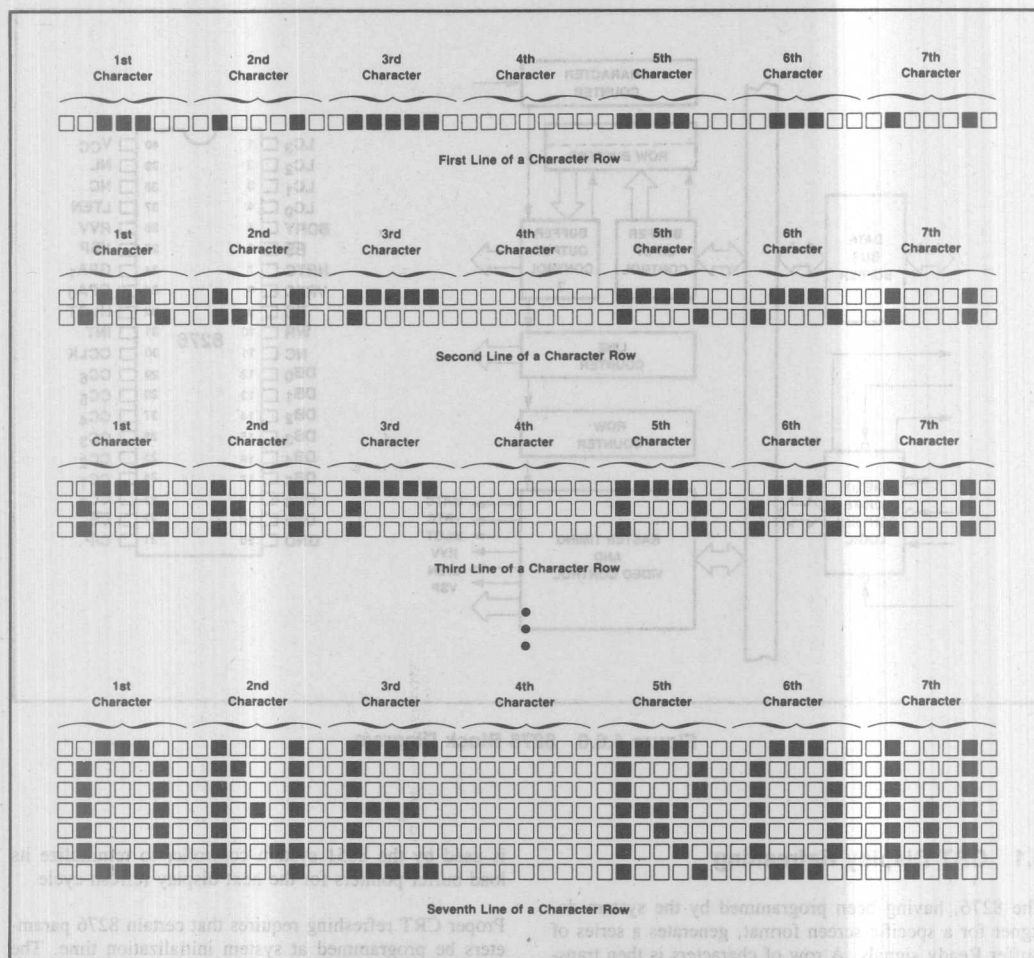


Figure 4.1.0 8276 Row Display

4.2 CRT Timing

The 8276 provides two timing outputs for controlling the CRT. The Horizontal Retrace Timing and Control (HRTC) and Vertical Retrace Timing and Control (VRTC) signals are used for synchronizing the CRT horizontal and vertical oscillators. A third output, VSP (Video Suppress), provides a signal to the dot timing logic to blank the video signal during the horizontal and vertical retraces. LTEN (Light Enable) is used to provide the ability to force the

video output high regardless of the state of the VSP signal. This feature is used to place the cursor on the screen and to control attribute functions. RVV (Reverse Video) output, if enabled, will cause the system to invert its video output. The fifth timing signal output, HLG (highlight) allows the flexibility to increase the CRT beam intensity to a greater than normal level.

COMMAND	NO. OF PARAMETER BYTES	NOTES
RESET	4	Display format parameters required
START DISPLAY	0	DMA operation parameters included in command
STOP DISPLAY	0	—
RED LIGHT PEN	2	—
LOAD CURSOR	2	Cursor X, Y position parameters required
ENABLE INTERRUPT	0	—
DISABLE INTERRUPT	0	—
PRESET COUNTERS	0	Clears all internal counters

Figure 4.1.1 8276 Instruction Set

4.3 Special Functions

4.3.1 Special Codes

The 8276 recognizes four special codes that may be used to reduce memory, software, or system controller overhead. These characters are placed within the display memory by the system controller. The 8276 performs certain tasks when these codes are received in its row buffer memory.

- 1) **End of Row Code** — Activates VSP. VSP remains active until the end of the line is reached. While VSP is active the screen is blanked.
- 2) **End Of Row-Stop Buffer Loading Code** — Causes the Buffer Ready control logic to stop requesting buffer transfers for the rest of the row. It affects the display the same as End of Row Code.
- 3) **End Of Screen Code** — Activates VSP. VSP remains active until the end of the frame is reached.

4) **End Of Screen-Stop Buffer Loading Code** — Causes the Buffer Ready control logic to stop requesting buffer transfers until the end of the frame is reached. It affects the display the same way as the End of Screen code.

4.3.4 Programmable Buffer Loading Control

The 8276 can be programmed to request 1, 2, 4, or 8 characters per Buffer load. The interval between loads is also programmable. This allows the designer the flexibility to tailor the buffer transfer overhead to fit the system needs.

Each scan line requires 63.5 microseconds. A character line consists of 10 scan lines and takes 635 microseconds to form. The 8276 row buffer must be filled within the 635 microseconds or an under run condition will occur within the 8276 causing the screen to be blanked until the next vertical retrace. This blanking will be seen as a flicker in the display.

5.0 DESIGN BACKGROUND

A fully functional, microcontroller-based CRT terminal was designed and constructed using the 8051 and the 8276. The terminal has many of the functions that are found in commercially available low cost terminals. Sophisticated features such as programmable keys can be added easily with modest amounts of software.

The 8051's functions in this application note include: up to 9.6K BAUD full duplex serial transmission; decoding special messages sent from the host computer; scanning, debouncing, and decoding a full function keyboard; writing to the 8276 row buffer from the display RAM without the need for a DMA controller; and scrolling the display.

The 8276 CRT controller's functions include: presenting the data to the character generator; providing the timing signals needed for horizontal and vertical retrace; and providing blanking and video information.

5.1 Design Philosophy

Since the device count relates to costs, size, and reliability of a system, arriving at a minimum device count without degrading the performance was a driving force for this application note. LSI devices were used where possible to maintain a low chip count and to make the design cycle as short as possible.

PL/M-51 was chosen to generate the majority of the software for this application because it models the human thought process more closely than assembly language. Consequently it is easier and faster to write programs using PL/M-51 and the code is more likely to be correct because less chance exists to introduce errors.

PL/M-51 programs are easier to read and follow than assembly language programs, and thus are easier to modify and customize to the end user's application. PL/M-51 also offers lower development and maintenance costs than assembly language programming.

PL/M-51 does have a few drawbacks. It is not as efficient in code generation relative to assembly language and thus may also run slower.

This application note uses the 8051's interrupts to control the servicing of the various peripherals. The speed of the main program is less critical if interrupts are used. In the last two application notes on terminal controllers, a criterion of the system was the time required for receiving an incoming serial byte, decoding it, performing the function requested, scanning the keyboard, debouncing the keys, and transmitting the decoded ASCII code must be less than the vertical refresh time. Using the 8051 and its interrupts makes this time constraint irrelevant.

5.2 System Target Specifications

The design specifications for the CRT terminal design is as follows:

Display Format

- 80 characters/display row
- 25 display lines

Character Format

- 5 × 7 character contained within a 7 × 10 frame
- First and seventh columns blanked
- Ninth line cursor position
- Programmable delay blinking underline cursor

Control Characters Recognized

- Backspace
- Linefeed
- Carriage Return
- Form Feed

Escape Sequences Recognized

- ESC A, Cursor up
- ESC B, Cursor down
- ESC C, Cursor right
- ESC D, Cursor left
- ESC E, Clear screen
- ESC F, Move addressable cursor
- ESC H, Home cursor
- ESC J, Erase from cursor to the end the screen
- ESC K, Erase the current line

Characters Displayed

- 96 ASCII Alphanumeric Characters

Characters Transmitted

- 96 ASCII Alphanumeric Characters
- ASCII Control Character Set
- ASCII Escape Sequence Set
- Auto Repeat

Display Memory

- 2K × 8 static RAM

Data Rate

- Variable rate from 150 to 9600 BAUD

CRT Monitor

- Ball Bros TV-12, 12MHZ Black and White

Keyboard

- Any standard undecoded keyboard (2 key lock-out)
- Any standard decoded keyboard with output enable pin
- Any standard decoded serial keyboard up to 150 BAUD

Scrolling Capability

Compatible With Wordstar

6.0 SYSTEM DESCRIPTION

A block diagram of the CRT terminal is shown in figure 6.0.0. The diagram shows only the essential system features. A detailed schematic of the CRT terminal is contained in the Appendix 7.1.

The "brains" of the CRT terminal is the 8051 microcontroller. The 8276 is the CRT controller in the system, and a 2716 EPROM is used as the character generator. To handle the high speed portion of the CRT, the 8276 is surrounded by a handful of TTL devices. A 2K × 8 static RAM was used as the display memory.

Following the system reset, the 8276 is initialized for cursor type, number of characters per line, number of lines, and character size. The display RAM is initialized to all "spaces" (ASCII 20H). The 8051 then writes the "start display" command to the 8276. The local/line input is sampled to determine the terminal mode. If the terminal is on-line, the BAUD rate switches are read and the serial port is set up for full duplex UART mode. The processor then is put into a loop waiting to service the serial port fifo or the 8276.

The serial port is programmed to have the highest priority interrupt. If the serial port generates an interrupt, the processor reads the buffer, puts the character in a generated fifo that resides in the 8051's internal RAM, increments the fifo pointer, sets the serial interrupt flag and returns.

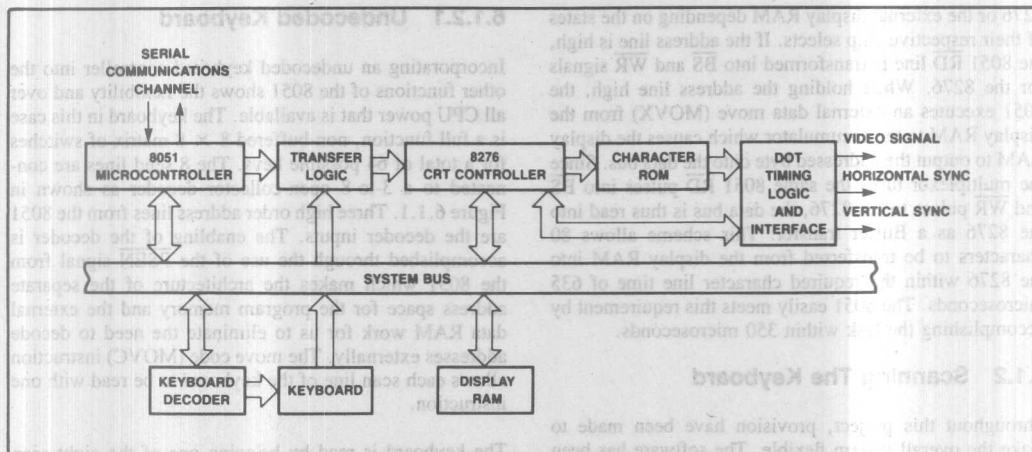


Figure 6.0.0 CRT Terminal Controller Block Diagram

The main program determines if it is a displayable character, a Control word or an ESC sequence and either puts the character in the display buffer or executes the appropriate command sent from the host computer.

If the 8276 needs servicing, the 8051 fills the row buffer for the CRT display's next line. If the 8276 generates a vertical retrace interrupt, the buffer pointers are reloaded with the display memory location that corresponds to the first character of the first display line on the CRT. The vertical retrace also signals the processor to read the keyboard for a key closure.

6.1 Hardware Description

The following section describes the unique characteristics of this design.

6.1.1 Peripheral Address Map

The display RAM, 8276 registers, and the 8276 row buffers are memory mapped into the external data RAM address area. The addresses are as follows:

Read and Write External Display RAM —	Address 1000H to 17CFH
Write to 8276 row buffers from Display RAM —	Address 1800H to 1FCFH
Write to 8276 Command Register (CREG) —	Address 0001H
Write to 8276 Parameter Register (PREG) —	Address 0000H
Read from 8276 Status Register (SREG) —	Address 0001H

Three general cases can be explored; reading and writing the display RAM, writing to the 8276 row buffers, and reading and writing the 8276's control registers.

As mentioned previously the 8051 fills the 8276 row buffer without the need of a DMA controller. This is accomplished by using a Quad 2-input multiplexor (Figure 6.1.0) as the transfer logic shown in the block diagram. The address line, P2.3, is used to select either of the two inputs. When the address line is low the RD and WR lines perform their normal functions, that is read and write the

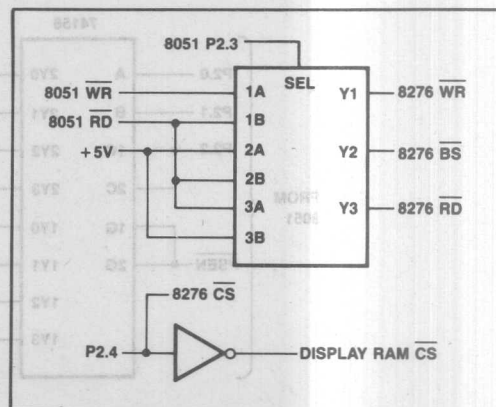


Figure 6.1.0
Simplified Version Of The Transfer Logic

closed, the data bus line is connected through the switch to the low output of the decoder and one of the data bus lines will be read as a 0. By knowing which scan line detected a key closure and which data bus line was low, the ASCII code for that key can easily be looked up in a matrix of constants. PL/M-51 has the ability to handle arrays and structured arrays, which makes the decoding of the keyboard a trivial task.

Since the Shift, Cap Lock, and Control keys may change the ASCII code for a particular key closure, it is essential to know the status of these pins while decoding the keyboard. The Shift, Cap Lock, and Control keys are therefore not scanned but are connected to the 8051 port pins where they can be tested for closure directly.

The 8 receive lines are connected to the data bus through germanium diodes which chosen for their low forward voltage drop. The diodes keep the keyboard from interfering with the data bus during the times the keyboard is not being read. The circuit consisting of the 3-to-8 decoder and the diodes also offers some protection to the 8051 from possible Electrostatic Discharge (ESD) damage that could be transmitted through the keyboard.

6.1.2.2 Decoded Keyboard

A decoded keyboard can easily be connected to the system as shown in Figure 6.1.2. Reading the keyboard can be evoked either by interrupts or by software polling.

The software to periodically read a decoded keyboard was not written for this application note but can be accomplished with one or two PL/M-51 statements in the READER routine.

A much more interesting approach would be to have the servicing of the keyboard be interrupt driven. An additional external interrupt is created by configuring timer/counter 0 into an event counter. The event counter is

initialized with the maximum count. The keyboard controller would inform the 8051 that a valid key has been depressed by pulling the input pin T0 low. This would overflow the event counter, thus causing an interrupt. The interrupt routine would simply use a MOVX (PSEN is connected to the output enable pin of the keyboard controller) to read the contents of the keyboard controller onto the data bus, reinitialize the counter to the maximum count and return from the interrupt.

6.1.2.3 Serial Decoded Keyboard

The use of detachable keyboards has become popular among the manufacturers of keyboards and personal computers. This terminal has provisions to use such a keyboard.

The keyboard controller would scan the keyboard, debounce the key and send back the ASCII code for that key closure. The message would be in an asynchronous serial format.

The flowchart for a software serial port is shown in Figure 6.1.3. An additional external interrupt is created as discussed for the decoded keyboard but the use in this case would be to detect a start bit. Once the beginning of the start bit has been detected, the timer/counter 0 is configured to become a timer. The timer is initialized to cause an interrupt one-half bit time after the beginning of the start bit. This is to validate the start bit. Once the start bit is validated, the timer is initialized with a value to cause an interrupt one bit time later to read the first data bit. This process of interrupting to read a data bit is repeated until all eight data bits have been received. After all 8 data bits are read, the software serial port is read once more to detect if a stop bit is present. If the stop bit is not present, an error flag is set, all pointers and flags are reset to their initial values, and the timer/counter is reconfigured to an event counter to detect the next start bit. If the stop bit is present, a valid flag is set and the flags and counter are reset as previously discussed.

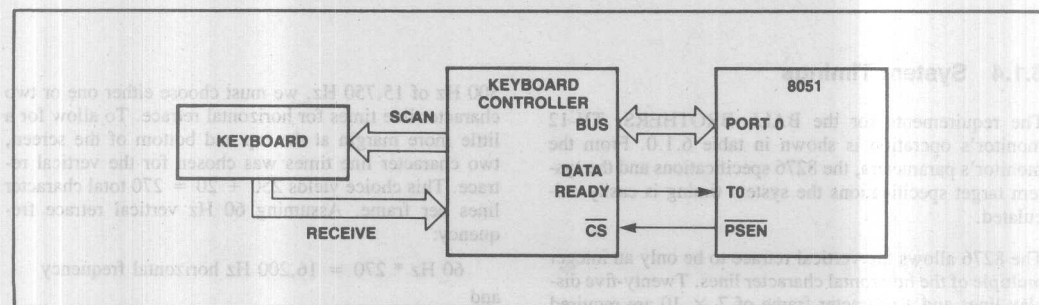


Figure 6.1.2 Using A Decoded Keyboard

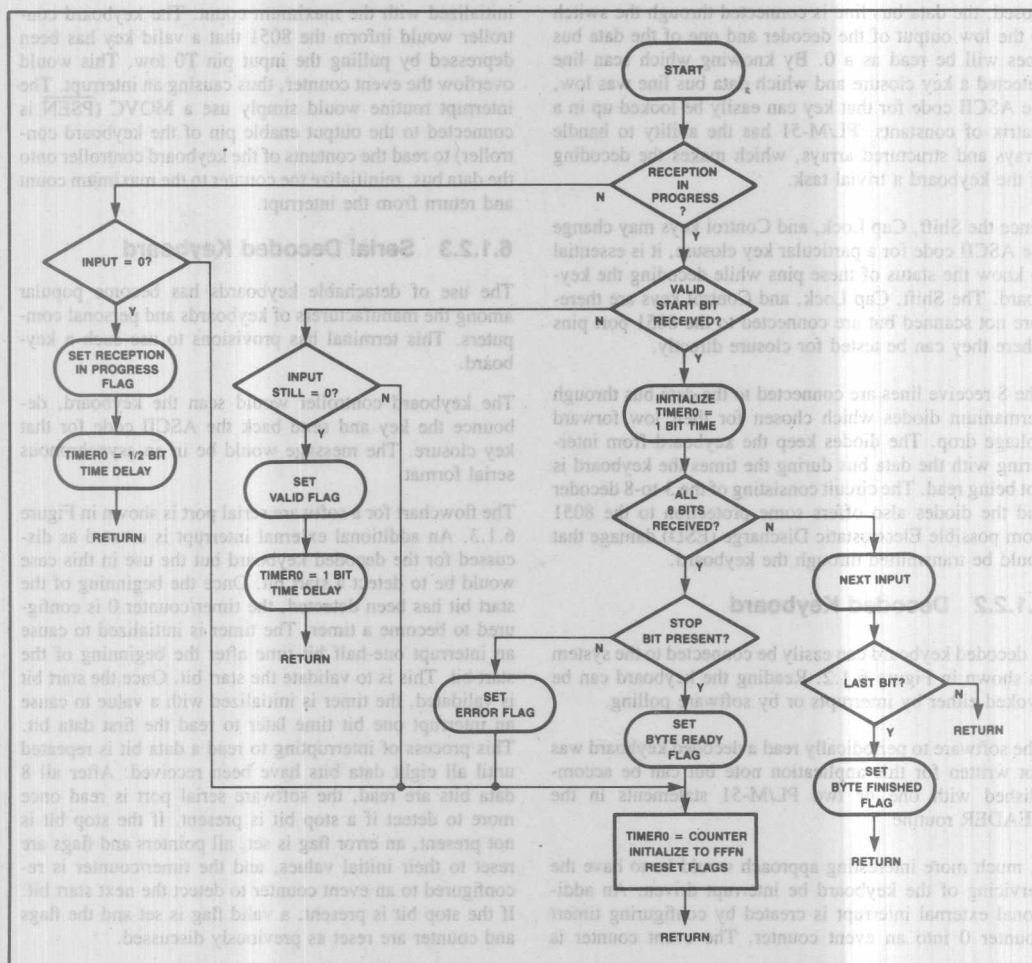


Figure 6.1.3 Flowchart for the Software Serial Port

6.1.4 System Timings

The requirements for the BALL BROTHERS, TV-12 monitor's operation is shown in table 6.1.0. From the monitor's parameters, the 8276 specifications and the system target specifications the system timing is easily calculated.

The 8276 allows the vertical retrace to be only an integer multiple of the horizontal character lines. Twenty-five display lines and a character frame of 7×10 are required from the target specification which will require 250 horizontal lines. If the horizontal frequency is to be within

500 Hz of 15,750 Hz, we must choose either one or two character line times for horizontal retrace. To allow for a little more margin at the top and bottom of the screen, two character line times was chosen for the vertical retrace. This choice yields $250 + 20 = 270$ total character lines per frame. Assuming 60 Hz vertical retrace frequency:

60 Hz * 270 = 16,200 Hz horizontal frequency
and

$$1/16,200 \text{ Hz} * 20 \text{ horizontal sync times} = 1.2345 \text{ milliseconds}$$

Table 6.1.0 CRT Monitor's Operational Requirements

PARAMETER	RANGE
Vertical Blanking Time (VRTC)	800 μ sec nominal
Vertical Drive Pulsewidth	$300 \mu\text{sec} \leq \text{PW} \leq 1.4 \text{ ms}$
Horizontal Blanking Time (HRTC)	11 μ sec nominal
Horizontal Drive Pulsewidth	$25 \mu\text{sec} \leq \text{PW} \leq 30 \mu\text{sec}$
Horizontal Repetition Rate	$15,750 \pm 500 \text{ pps}$

The 1.2345 milliseconds of retrace time meets the nominal VRTC and vertical drive pulse width time of 1.3mSec to 1.4mSec for the Ball monitor.

The next parameter to find is the horizontal retrace time which is wholly dependent on the monitor used. Usually it lies between 15 and 30 percent of the total horizontal line time.

Since most designs display a fixed number of characters per line it is useful to express the horizontal retrace time as a given number of character times. In this design, 80 characters are displayed, and it was experimentally found that 20 character times for the horizontal retrace gave the best results. It should be noted if too much time was given for retrace, there would be less time to display the characters and the display would not fill out the screen. Conversely, if not enough time is given for retrace, the characters would seem to run off the screen.

One hundred character times per complete horizontal line means that each character needs:

$(1/16,200 \text{ Hz}) / 100 \text{ character times} = 617.3 \text{ nanoseconds}$

If we multiply the 20 character times needed to retrace by 617.3 nanoseconds needed for each character, we find 12.345 microseconds are allocated for retrace. This value falls short of the 25 to 30 microseconds required by the horizontal drive of the Ball monitor. To correct for this, a 74LS123 one-shot was used to extend the horizontal drive pulse width.

The dot clock frequency is easy to calculate now that we know the horizontal frequency. Since each character is formed by seven dots in the horizontal axis, the dot clock period would be the character clock (617.3 nanoseconds) divided by the 7 which is equal to 11.34 MHz. The basic dot timing and CRT timing are shown in the Appendix.

6.2 Software Description

6.2.1 Software Overview

The software for this application was written in a "foreground-background" format. The background programs are all interrupt driven and are written in assembly language due to time constraints. The foreground programs are for the most part written in PL/M-51 to ease the programming effort. A number of subroutines are written in assembly language due to time constraints during execution. Subroutines such as clearing display lines, clearing the screen, and scanning the keyboard require a great deal of 16 bit adds and compares and would execute much slower and would require more code space if written in PL/M-51. The background and foreground programs talk to each other through a set of flags. For example, the PL/M-51 foreground program tests "SERIAL\$INT" to determine if a serial port interrupt had occurred and a character is waiting to be processed.

6.2.2 The Background Program

Two interrupt driven routines, VERT and BUFFER, (see Fig. 6.2.0) request service every 16.67 milliseconds and 617 microseconds respectively. VERT's request comes during the last character row of the display screen. This routine resets the buffer pointers to the first CRT display line in the display memory. VERT is also used as a time base for the foreground program. VERT sets the flag, SCAN, to tell the foreground program (PL/M-51) that it is time to scan the Keyboard. VERT also increments a counter used for the delay between transmitting characters in the AUTOSREPEAT routine.

The BUFFER routine is executed once per character row. BUFFER uses the multiplexor discussed earlier to fill the 8276's row buffer by executing 80 external data moves and incrementing the Data Pointer between each move.

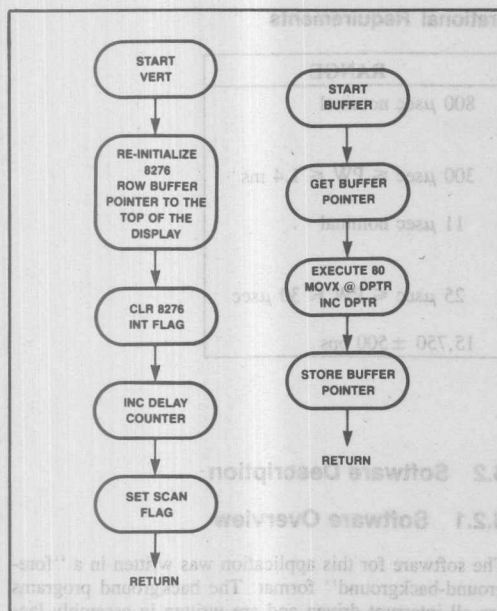


Figure 6.2.0 Flowcharts For VERT and BUFFER Routine

The MOVX reads the display RAM and writes the character into the row buffer during the same instruction.

SERBUF is an interrupt driven routine that is executed each time a character is received or transmitted through the on-chip serial port. The routine first checks if the interrupt was caused by the transmit side of the serial port, signaling that the transmitter is ready to accept another character. If the transmitter caused the interrupt, the flag "TRANSMIT\$INT" is set which is checked by the foreground program before putting a character in the buffer for transmission.

If the receiver caused the interrupt, the input buffer on the serial port is read and fed into the fifo that has been manufactured in the internal RAM and increments the fifo pointer "FIFO." The flag "SERIAL\$INT" is then set, telling the foreground program that there is a character in the fifo to be processed. If the read character is an ESC character, the flag "ESCSEQ" is set to tell the foreground program that an escape sequence is in the process of being received.

6.2.3 The Foreground Program

The foreground program is documented in the Appendix. The foreground program starts off by initializing the 8276

as discussed earlier. After all variables and flags are initialized, the processor is put into a loop waiting for either VERT to set SCAN so the program can scan the keyboard, or for the serial port to set SERIAL\$INT so the program can process the incoming character.

The vertical retrace is used to time the delay between keyboard scans. When VERT gets set, the assembly language routine READER is called. READER scans the keyboard, writing each scan into RAM to be processed later. READER controls two flags, KEY0 and SAME. KEY0 is set when all 8 scans determine that no key is pressed. SAME is set when the same key that was pressed last time the keyboard was read is still pressed.

After READER returns execution to the main program, the flags are tested. If the KEY0 flag is set the main program goes back to the loop waiting for the vertical retrace or a serial port interrupt to occur. If the SAME flag is set the main program knows that the closed key has been debounced and decoded so it sends the already known ASCII code to the AUTO\$REPEAT routine which determines if that character should be transmitted or not.

If KEY0 and SAME are not set, signifying that a key is pressed but it is not the same key as before, the foreground program determines if the results from the scan are valid. First all eight scans are checked to see if only one key was closed. If only one key is closed, the ASCII code is determined, modified if necessary by the Shift, Cap Lock, or Control keys. The NEWSKEY and VALID flags are then set. The next time READER is called, if the same key is still pressed, the SAME flag will be set, causing the AUTO\$REPEAT subroutine to be called as just discussed. Since the keyboard is read during the vertical retrace, 16.67 milliseconds has elapsed between the detection of the pressed key and reverifying that the key is still pressed before transmitting it, thus effectively debouncing the key.

The AUTO\$REPEAT routine is written to transmit any key that the NEWSKEY flag is set for. The counter that is incremented each time the vertical refresh interrupt is serviced causes a programmable delay between the first transmission and subsequent auto repeat transmission. Once the NEWSKEY character is sent, the counter is initialized. Each time the AUTO\$REPEAT routine is called, the counter is checked. Only when the counter overflows will the next character be transmitted. After the initial delay, a character will be transmitted every other time the routine is called as long as the key remains pressed.

6.2.3.1 Handling Incoming Serial Data

One of the criteria for this application note was to make the software less time dependent. By creating a fifo to store incoming characters until the 8051 has time to pro-

cess them, software timing becomes less critical. This application note uses up to 8 levels of the fifo at 9.2KBAUD, and 1 level at 4.8KBAUD and lower. As discussed earlier, the interrupt service routine for the serial port uses the fifo to store incoming data, increments the fifo pointer, "FIFO", and sets SERIAL\$INT to tell the main program that the fifo needs servicing. Once the main program detects that SERIAL\$INT is set the routine DECIPHER is executed.

DECIPHER has three separate blocks; a block for decoding displayable characters, a block for processing Escape sequences, and a block for processing Control codes. Each block works on the fifo independently. Before exiting a block, the contents of the fifo are shifted up by the amount of characters that were processed in that particular block. The shifting of the characters insures that the beginning of the fifo contains the next character to be processed. FIFO is then decremented by the number of characters processed.

Let's look at this process more closely. Figure 6.2.1-A shows a representation of a fifo containing 5 characters. The first three characters in the fifo contain displayable characters, A, B, and C respectively with the last two characters being an ESC sequence for moving the cursor up one line (ESC A) and FIFO points to the next available location to be filled by the serial port interrupt routine, in this case, 5.

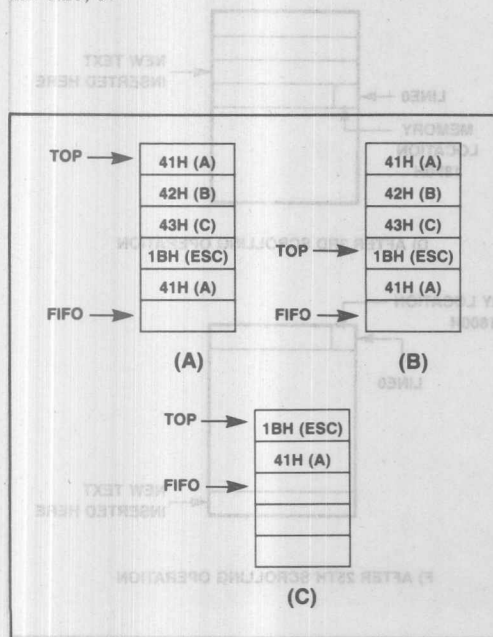


FIGURE 6.2.1 FIFO

When DECIPHER is executed, the first block begins looking at the first character of the fifo for a displayable character. If the character is displayable, it is placed into the display RAM and the software pointer "TOP" that points to the character that is being processed is incremented to the next character. The character is then looked at to see if it too is displayable and if it is, it's placed in the display RAM. The process of checking for displayable characters is continued until either the fifo is empty or a non-displayable character is detected. In our example, three characters are placed into the display RAM before a non-displayable character is detected. At this point the fifo looks like figure 6.2.1-B.

Before entering the next block, the remaining contents of the fifo between TOP, that is now pointing to 1BH and (FIFO-1) are moved up in the fifo by the amount of characters processed, in this example three. TOP is reset to 0 and FIFO is decremented by 3. The serial port interrupt is inhibited during the time the contents of the fifo and the pointers are being manipulated. The fifo now looks like figure 6.2.1-C.

The execution is now passed to the next block that processes ESC sequences. The first location of the fifo is examined to see if it is an ESC character (1BH). If not, the execution is passed to the next block of DECIPHER that processes Control codes. In this case the fifo does contain an ESC code. The flag ESC\$SEQ is checked to see if the 8051 is in the process of receiving an ESC sequence thus signifying that the next byte of the sequence has not been received yet. If the ESC\$SEQ is not set, the next character in the fifo is checked for a valid escape code and the proper subroutine is then called. The fifo contents are then shifted as discussed for the previous block. Due to the length of time that is needed to execute an ESC code sequence or a Control code, only one ESC code and/or Control code can be processed each time DECIPHER is executed.

If at the end of the DECIPHER routine, FIFO contains a 0, the flag SER\$INT is reset. If SER\$INT remains set, DECIPHER will be executed immediately after returning to the main program if SCAN had not been set during the execution of the DECIPHER routine, otherwise DECIPHER will be called after the keyboard is read.

6.2.4 Memory Pointers and Scrolling

The cursor always points to the next location in display memory to be filled. Each time a character is placed in the display memory, the cursor position needs to be tested to determine if the cursor should be incremented to the beginning of the next line of the display or simply moved to the next position on the current display line. The cursor position pointers are then updated in both the 8276 and the internal registers in the 8051.

When the 2000th character is entered into the display memory, a full display page has been reached signaling the need for the display to scroll. The memory pointer that points to the display memory that contains the first character of the first display line, LINE0, prior to scrolling contains 1800H which is the starting address of the display memory. Each scrolling operation adds 80 (50H) to LINE0 which will now point to the following row in memory as shown in figure 6.2.2-B. LINE0 is used during the vertical

refresh routine to re-initialize the pointers associated with filling the 8276 row buffers.

The display memory locations that were the first line of the CRT display now becomes the last line of the CRT display. Incoming characters are now entered into the display memory starting with 1800H, which is now the first character of the last line of the display screen.

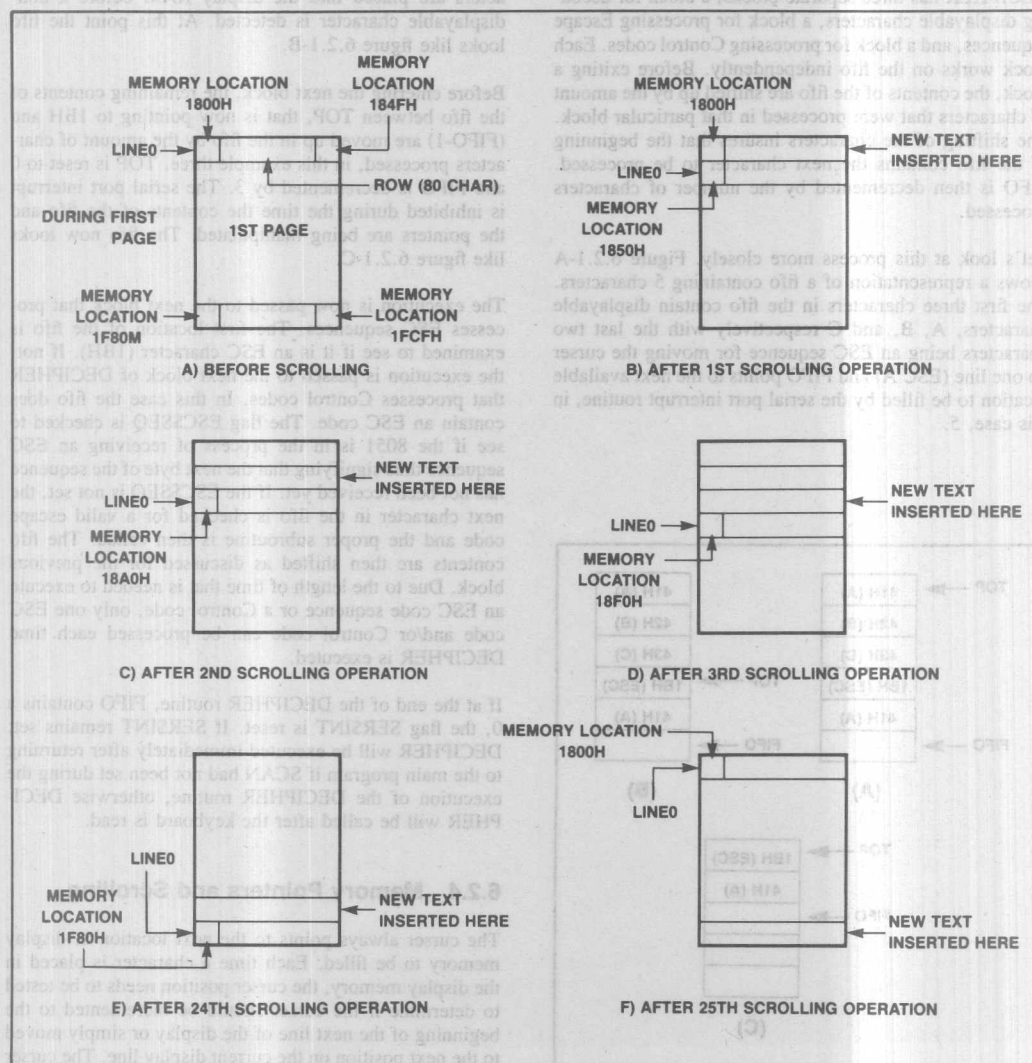


Figure 6.2.2 Pointer Manipulation During Scrolling

6.2.5 Software Timing

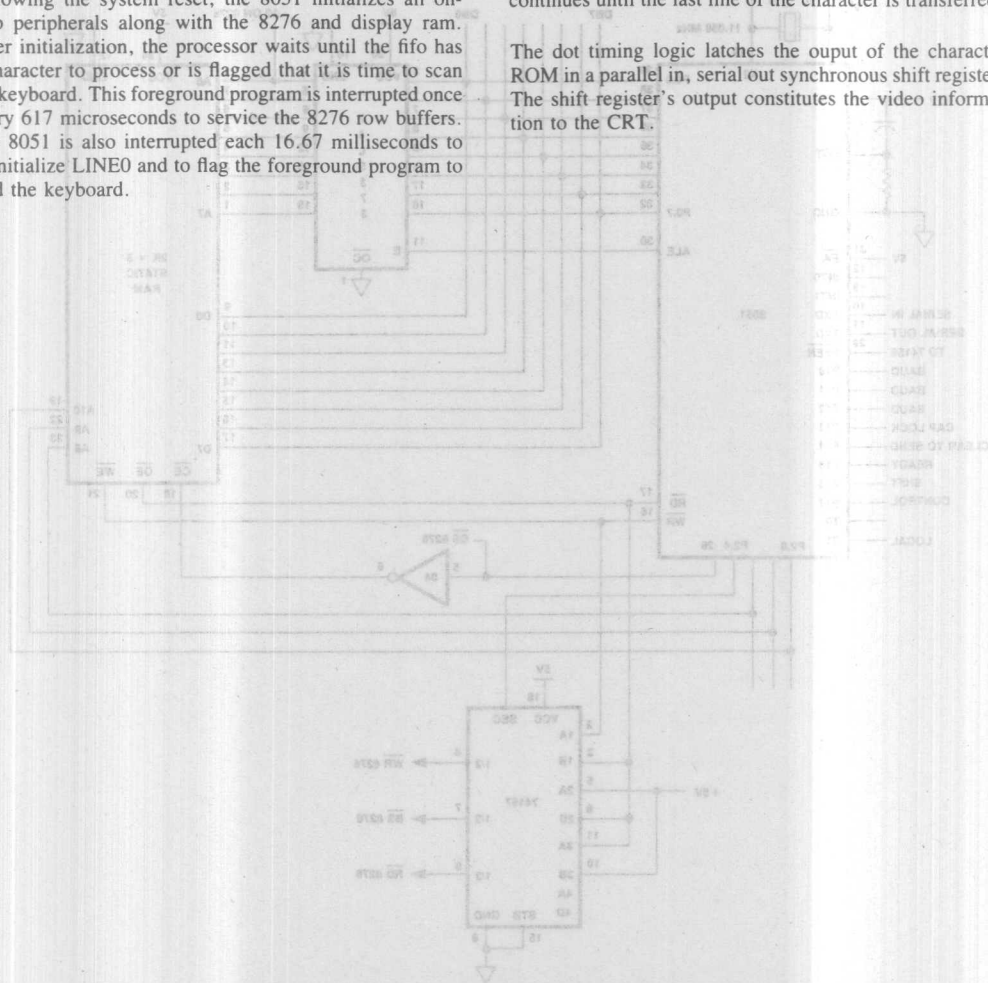
The use of interrupts to tie the operation of the foreground program to the real-time events of the background program has made the software timing non-critical for this system.

6.3 System Operation

Following the system reset, the 8051 initializes all on-chip peripherals along with the 8276 and display ram. After initialization, the processor waits until the fifo has a character to process or is flagged that it is time to scan the keyboard. This foreground program is interrupted once every 617 microseconds to service the 8276 row buffers. The 8051 is also interrupted each 16.67 milliseconds to re-initialize LINE0 and to flag the foreground program to read the keyboard.

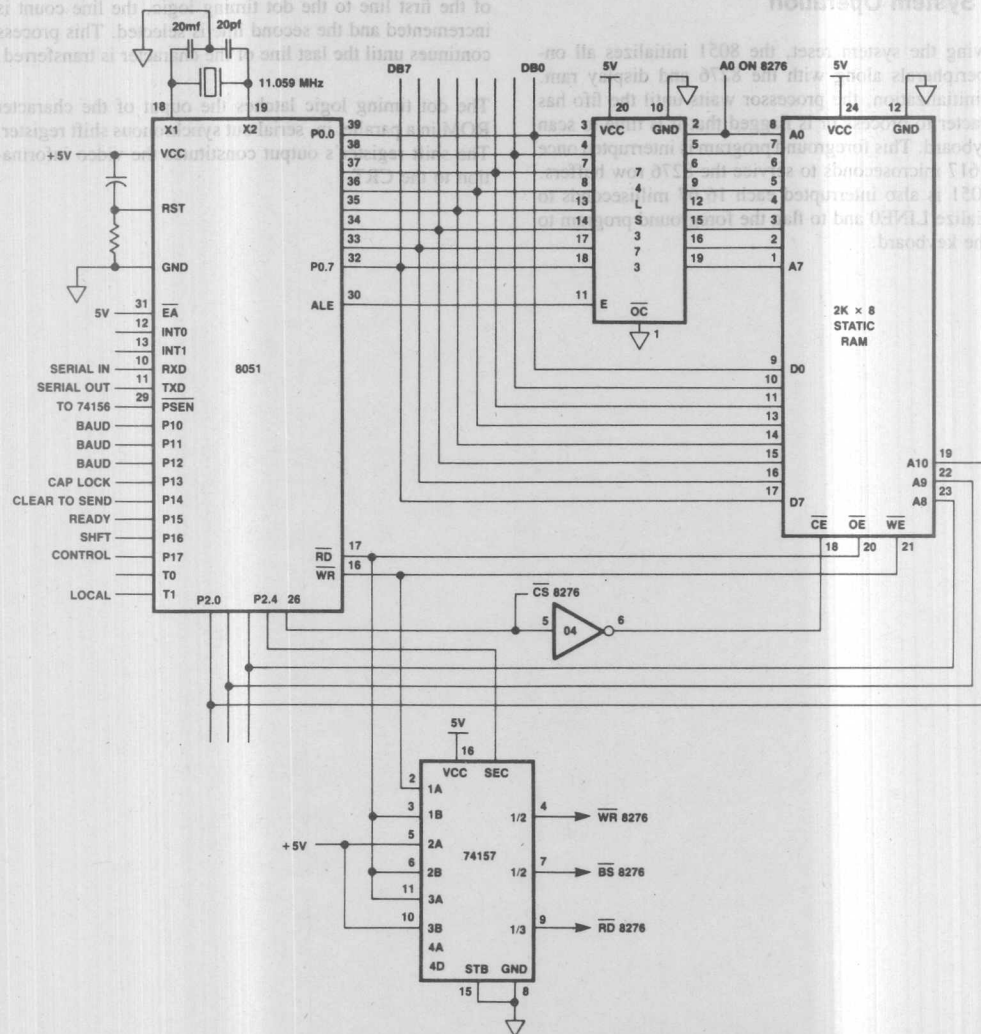
As discussed earlier, a special technique of rapidly moving the contents of the display RAM to the 8276 row buffers without the need of a DMA device was employed. The characters are then synchronously transferred to the character generator via CC0-CC6 and LC0-LC2 which are used to display one line at a time. Following the transfer of the first line to the dot timing logic, the line count is incremented and the second line is selected. This process continues until the last line of the character is transferred.

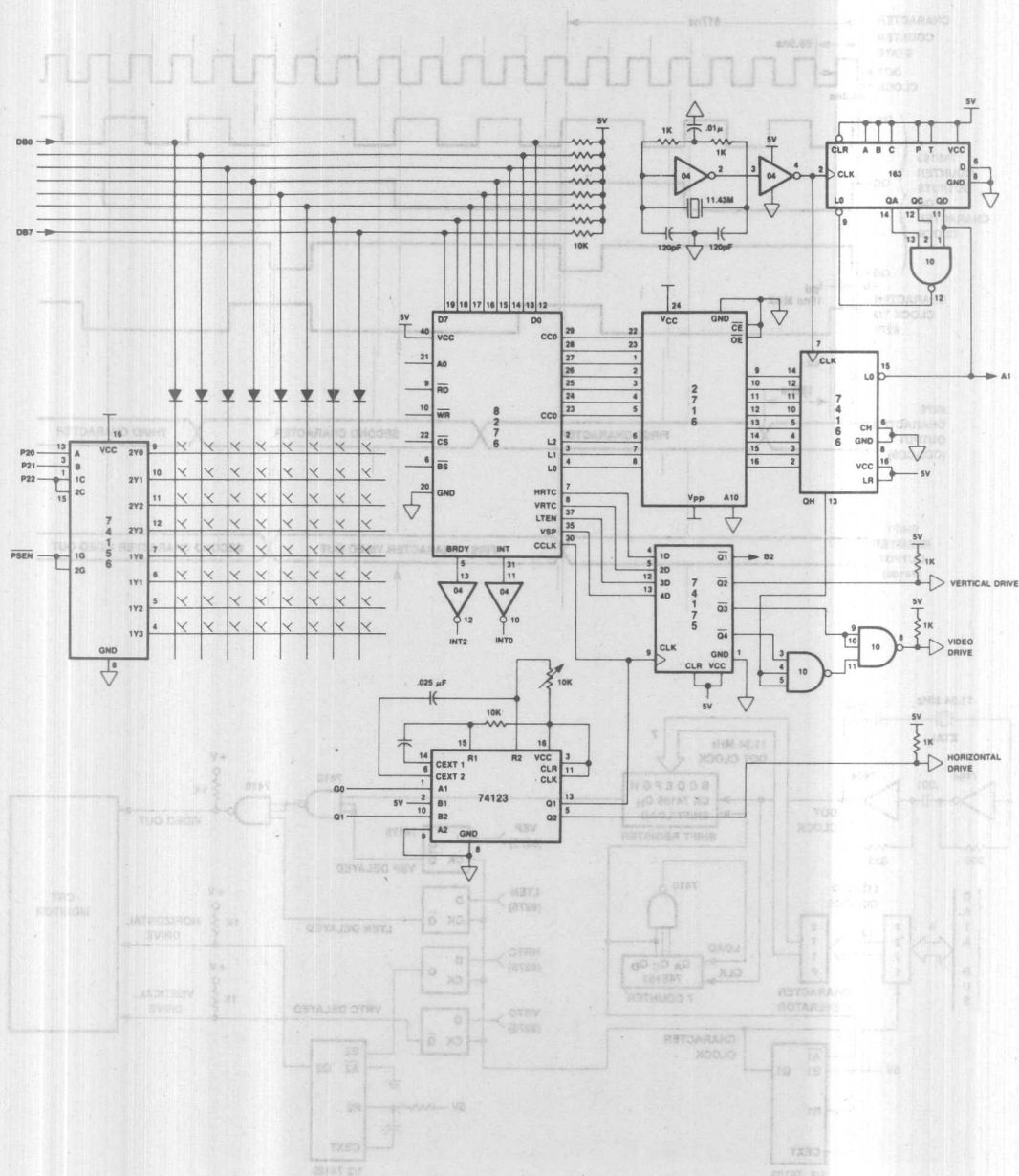
The dot timing logic latches the output of the character ROM in a parallel in, serial out synchronous shift register. The shift register's output constitutes the video information to the CRT.



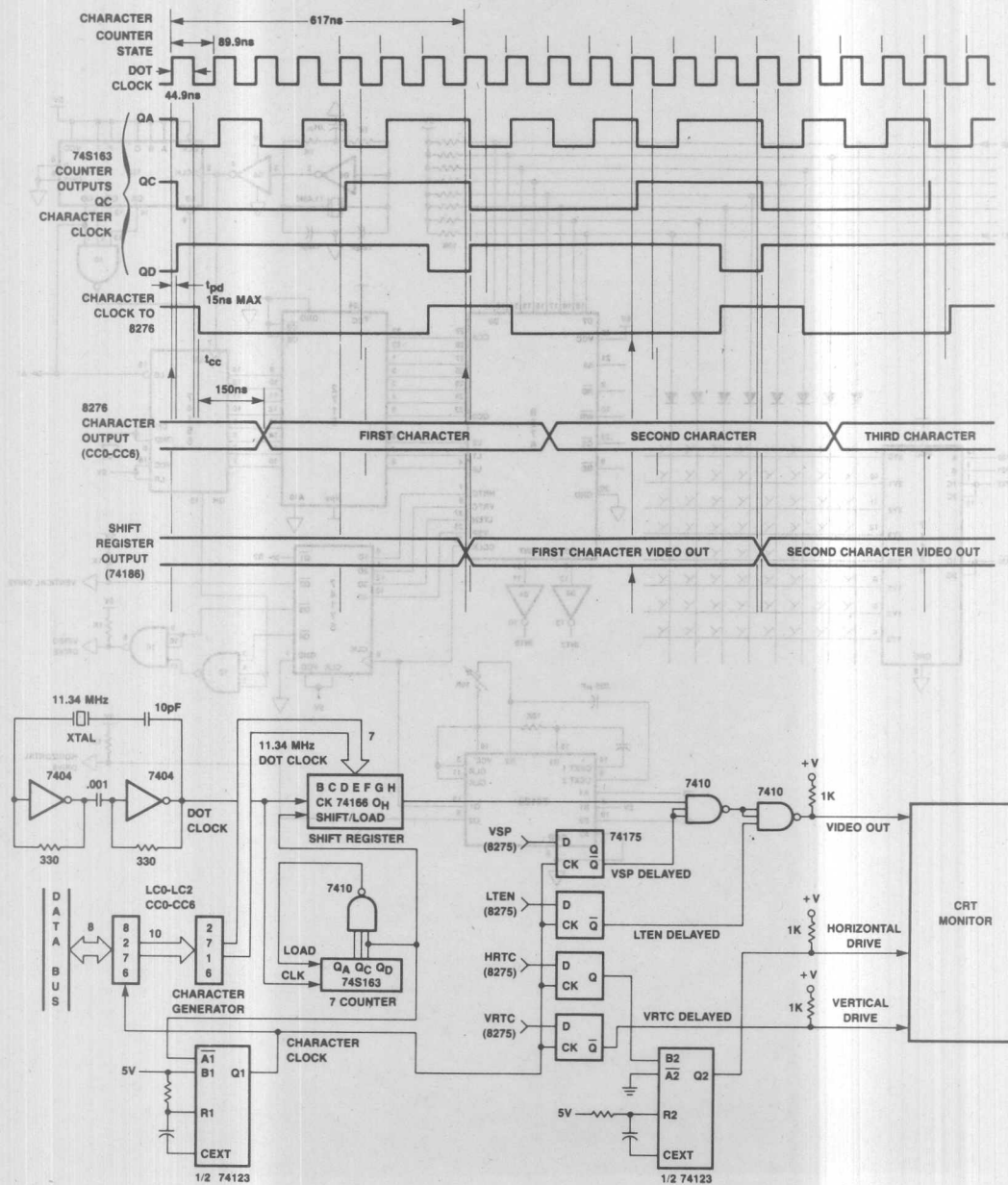
Appendix 7.1 CRT Schematics

As discussed earlier, the contents of the display RAM to the 8276 row buffers without the need of a DMA device was employed. The characters are then synchronously transferred to the character generator via CCO-CC0 and CCO-1-CC1 which are used to display one line at a time. Following the transfer of the first line to the dot timing logic, the line count is incremented and the second line is transferred. This process continues until the last line is transferred.

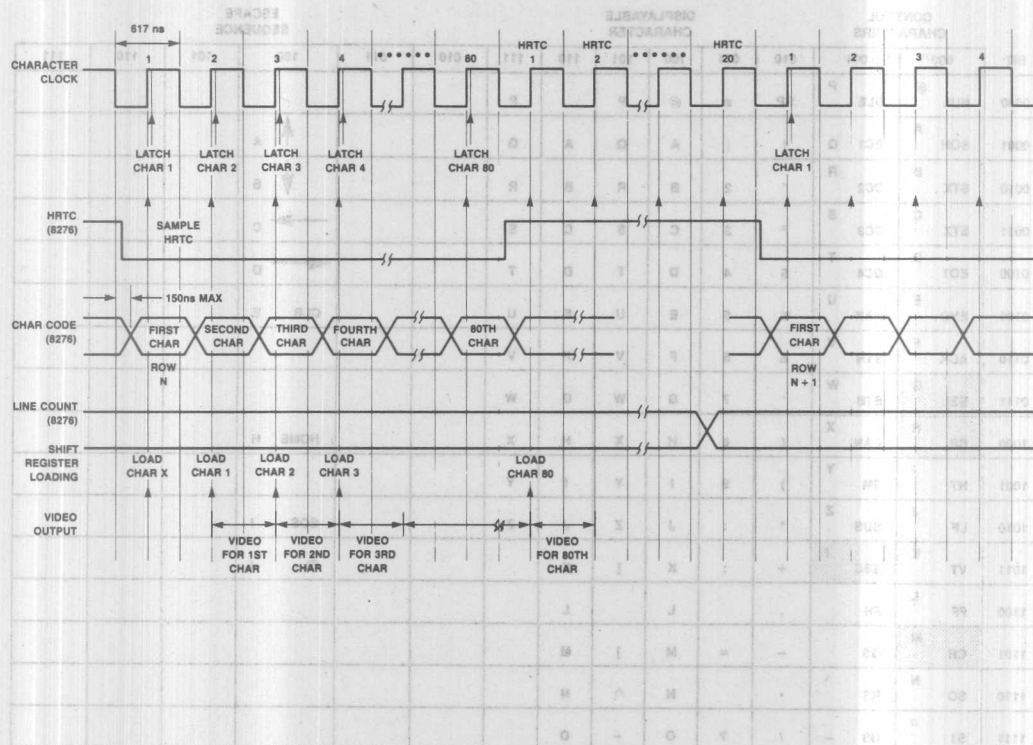




Appendix 7.2 Dot Timing



Appendix 7.3 CRT System Timing



Appendix 7.4 Escape/Control/Display Character Summary

CONTROL CHARACTERS					DISPLAYABLE CHARACTER					ESCAPE SEQUENCE				
BIT	000	001	010	011	100	101	110	111	010	011	100	101	110	111
0000	NUL @	DLE P	SP		@	P		P						
0001	SOH A	DC1 Q	!		A	Q	A	Q			↑ A			
0010	STX B	DC2 R	"	2	B	R	B	R			↓ B			
0011	ETX C	DC3 S	=	3	C	S	C	S			→ C			
0100	EOT D	DC4 T	\$	4	D	T	D	T			← D			
0101	ENQ E	NAK U	%	5	E	U	E	U			CLR E			
0110	ACK F	SYN V	&	6	F	V	F	V						
0111	BEL G	ETB W	'	7	G	W	G	W						
1000	BS H	CAN X	(8	H	X	H	X			HOME H			
1001	HT I	EM Y)	9	I	Y	I	Y						
1010	LF J	SUB Z	*	:	J	Z	J	Z			EOS I			
1011	VT K	ESC	+	;	K	[K				EL J			
1100	FF L	FS	,		L		L							
1101	CR M	GS	-	=	M]	M							
1110	SO N	RS	.		N	^	N							
1111	SI O	US -	/	?	O	-	O							

NOTE: Shaded blocks — functions terminal will react to. Others can be generated but are ignored upon receipt.

Appendix 7.5 Character Generator

As previously mentioned, the character generator used in this terminal is a 2716 EPROM. A 1K by 8 device would have been sufficient since a 128 character 5 by 7 dot matrix only requires 8K of memory. A custom character set could have been stored in the second 1K bytes of the 2716. Any of the free I/O pins on the 8051 could have been used to switch between the character sets.

The three low-order line count outputs (LC0-LC2) from the 8276 are connected to the three low-order address lines of the character generator. The CC0-CC6 output lines are connected to the A3-A9 lines of the character generator.

The output of the character generator is loaded into the shift register. The serial output of the shift register is the video output to the CRT.

Let's assume that the letter "E" is to be displayed. The ASCII code for "E" (45H) is presented to the address lines A2-A9 of the character generator. The scan lines (LC0-LC2) will now count from 0 to seven to form the character as shown in Figure 7.5.0. The same procedure is used to form all 128 possible characters. For reference Appendix 7.6 contains the HEX dump of the character generator used in this terminal.

45H = 01000101
Address to Prom = 01000101
= 228H - 22FH
Depending on state of Scan
lines.

Character generator output

Rom Address	Rom Hex Output	Bit Output*
228H	3E	0 1 2 3 4 5 6 7
229H	02	
22AH	02	
22BH	0E	
22CH	02	
22DH	02	
22EH	3E	
22FH	00	

Bits 0, 6 and 7 are not used.

*note bit output is backward from convention.

Figure 7.5.0 Character Generator

Appendix 7.7 Composite Video

In this design it was assumed that the CRT monitor required a separate horizontal drive, vertical drive, and video input. Many monitors require a composite video signal. The schematic shown in Figure 7.7.0 illustrate how to generate a composite video from the output of the 8276.

The dual one-shots are used to provide a small delay and the proper horizontal and vertical pulse to the composite video monitor. The delay introduced in the horizontal and vertical timing is used to center the display. The 7486 is used to mix the vertical and horizontal retrace. Q1 mix the video and retrace signals along with providing the proper D.C. levels.

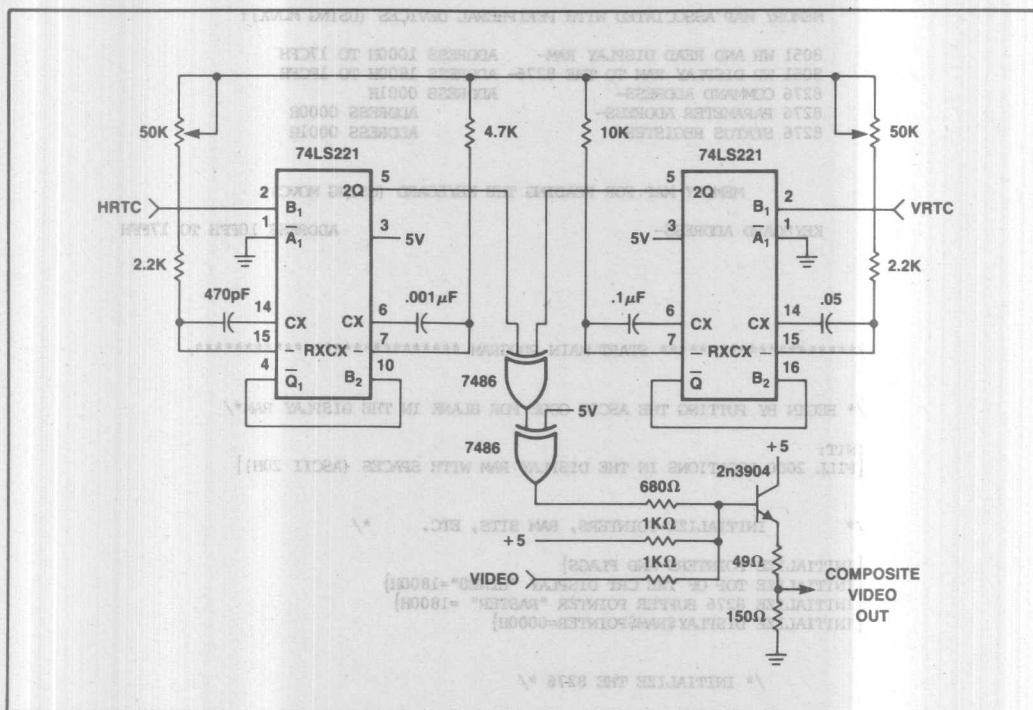


Figure 7.7.0 Composite Video

Appendix 7.8 Software Documentation

Appendix 7.7 Composite Video

```

*****
***** SOFTWARE DOCUMENTATION FOR THE 8051 *****
***** TERMINAL CONTROLLER APPLICATION NOTE *****
*****
*****
*****
*****

```

MEMORY MAP ASSOCIATED WITH PERIPHERAL DEVICES (USING MOVX):

```

8051 WR AND READ DISPLAY RAM- ADDRESS 1000H TO 17CFH
8051 WR DISPLAY RAM TO THE 8276- ADDRESS 1800H TO 1FCFH
8276 COMMAND ADDRESS- ADDRESS 0001H
8276 PARAMETER ADDRESS- ADDRESS 0000H
8276 STATUS REGISTER- ADDRESS 0001H

```

MEMORY MAP FOR READING THE KEYBOARD (USING MOVC):

```

KEYBOARD ADDRESS- ADDRESS 10FFH TO 17FFH

```

```

/***** START MAIN PROGRAM *****/

```

```

/* BEGIN BY PUTTING THE ASCII CODE FOR BLANK IN THE DISPLAY RAM*/

```

```

INIT:
{FILL 2000 LOCATIONS IN THE DISPLAY RAM WITH SPACES (ASCII 20H)}

```

```

/* INITIALIZE POINTERS, RAM BITS, ETC. */

```

```

{INITIALIZE POINTERS AND FLAGS}
{INITIALIZE TOP OF THE CRT DISPLAY "LINE0"=1800H}
{INITIALIZE 8276 BUFFER POINTER "RASTER" =1800H}
{INITIALIZE DISPLAY$RAM$POINTER=0000H}

```

```

/* INITIALIZE THE 8276 */

```

```

{RESET THE 8276}
{INITIALIZE 8276 TO 80 CHARACTER/ROW}
{INITIALIZE 8276 TO 25 ROWS PER FRAME}
{INITIALIZE 8276 TO 10 LINES PER ROW}
{INITIALIZE 8276 TO NON-BLINKING UNDERLINE CURSER}
{INITIALIZE CURSER TO HOME POSITION (00,00) (UPPER LEFT HAND CORNER)}
{START DISPLAY}
{ENABLE 8276 INTERRUPT}

```

```

/* SET UP 8051 INTERRUPTS AND PRIORITIES */

```

```

{SERIAL PORT HAS HIGHEST INTERRUPT PRIORITY}
{EXTERNAL INTERRUPTS ARE EDGE SENSITIVE}
{ENABLE EXTERNAL INTERRUPTS}

```



```

/*PROCEDURE SCANNER: THIS PROCEDURE SCANS THE KEYBOARD AND DETERMINES IF A
SINGLE VALID KEY HAS BEEN PUSHED. IF TRUE THEN THE ASCII EQUIVALENT
WILL BE TRANSMITTED TO THE HOST COMPUTER.*/

```

```

SCANNER:

```

```

{ENABLE 8051 GLOBAL INTERRUPT BIT}

```

```

/* PROGRAMMABLE DELAY FOR THE CURSER BLINK */

```

```

IF {30 VERTICAL RETRACE INTERRUPTS HAVE OCCURRED (CURSER$COUNT=1FH)} THEN

```

```

DO;

```

```

{COMPLEMENT CURSER$ON}

```

```

{CLEAR CURSER$COUNT}

```

```

IF {CURSER IS TO BE OFF (CURSER$ON=0)} THEN {MOVE CURSER OFF THE SCREEN}

```

```

CALL LOAD$CURSER;

```

```

END;

```

```

IF {THE LOCAL$LINE SWITCH HAS CHANGED STATE} THEN

```

```

DO;

```

```

IF {IN LOCAL MODE} THEN {DISABLE SERIAL PORT INTERRUPT}

```

```

ELSE CALL CHECK$BAUD$RATE;

```

```

END;

```

```

DO WHILE {IN BETWEEN VERTICAL REFRESHES}

```

```

IF {THE FIFO HAS A CHARACTER TO PROCESS (SERIAL$INT=1)} THEN CALL DECIPHER;

```

```

END;

```

```

CALL READER;

```

```

IF {THE PRESENT PRESSED KEY IS EQUAL TO THE LAST KEY PRESSED AND VALID=1} THEN

```

```

CALL AUTO$REPEAT;

```

```

ELSE

```

```

DO;

```

```

IF {A KEY IS PRESSED BUT NOT THE SAME ONE AS THE LAST KEYBOARD SCAN} THEN

```

```

DO;

```

```

IF {ONLY ONE KEY IS PRESSED} THEN

```

```

{GET THE ASCII CODE FOR IT}

```

```

{SET NEWSKEY AND VALID FLAGS}

```

```

ELSE {RESET VALID AND NEWSKEY FLAGS}

```

```

END;

```

```

ELSE {THE KEYBOARD MUST NOT HAVE A KEY PRESSED SO RESET VALID$KEY AND NEWSKEY FLAGS}

```

```

END;

```

```

GOTO SCANNER;

```

```

END;

```

```

/* PROCEDURE AUTO$REPEAT: THIS PROCEDURE WILL PERFORM AN AUTO REPEAT FUNCTION
BY TRANSMITTING A CHARACTER EVERY OTHER TIME THIS ROUTINE IS CALLED.
THE AUTO REPEAT FUNCTION IS ACTIVATED AFTER A FIXED DELAY PERIOD AFTER THE
FIRST CHARACTER IS SENT*/

```

```

AUTO$REPEAT:

```

```

IF {THE KEY PRESSED IS NEW (NEWSKEY=1)} THEN

```

```

DO;

```

```

{CLEAR THE DIVIDE BY TWO COUNTER "TRANSMIT$TOGGLE"}

```

```

{INITIALIZE THE DELAY COUNTER "TRANSMIT$COUNT" TO 0D0H}

```

```

CALL TRANSMIT;

```

```

{CLEAR NEWSKEY}

```

```

END;

```

```

/* FIRST CHARACTER */

```

```

ELSE
DO;
  IF (TRANSMIT$COUNT IS NOT EQUAL TO 0) THEN
  DO;
    {INCREMENT TRANSMIT$COUNT}
    IF TRANSMIT$COUNT=0FFH THEN
    DO;
      CALL TRANSMIT;
      {CLEAR TRANSMIT$COUNT}
    END;
  END;
END;
ELSE
DO;
  {TURN THE CURSER ON DURING THE AUTO REPEAT FUNCTION}
  IF TRANSMIT$TOGGLE = 1 THEN
  DO;
    CALL TRANSMIT;
    {COMPLEMENT TRANSMIT$TOGGLE}
  END;
END;
END AUTO$REPEAT;

/* PROCEDURE TRANSMIT- ONCE THE HOST COMPUTER SIGNALS THE 8051H BY BRINGING
THE CLEAR-TO-SEND LINE LOW, THE ASCII CHARACTER IS PUT INTO THE SERIAL PORT.*/

TRANSMIT:
PROCEDURE;
IF (THE TERMINAL IS ON-LINE) THEN
DO;
  {WAIT UNTIL THE CLEAR$TO$SEND LINE IS LOW AND UNTIL THE 8051 SERIAL PORT TX IS NOT BUSY (TRANSMIT$INT=1)}
  {TRANSMIT THE ASCII CODE}
  {CLEAR THE FLAG "TRANSMIT$INT". THE SERIAL PORT SERVICE ROUTINE WILL SET THE FLAG
  WHEN THE SERIAL PORT IS FINISHED TRANSMITTING}
END;
ELSE {THE TERMINAL IS IN THE LOCAL MODE}
DO;
  {PUT THE ASCII CODE IN THE FIFO}
  {INCREMENT THE FIFO POINTER}
  {SET SERIAL$INT}
END;
END TRANSMIT;

```

```

/*      PROCEDURE DECIPHER: THIS PROCEDURE DECODES THE HOST COMPUTER'S MESSAGES AND DETERMINES
      WHETHER IT IS A DISPLAYABLE CHARACTER, CONTROL SEQUENCE, OR AN ESCAPE SEQUENCE
      THE PROCEDURE THEN ACTS ACCORDINGLY */

```

```

DECIPHER:
START$DECIPHER:

```

```

VALID$RECEPTION=0;
DO WHILE {THE FIFO IS NOT EMPTY AND THE CHARACTER IS DISPLAYABLE}
  RECEIVE={ASCII CODE}
  CALL DISPLAY;
  {NEXT CHARACTER}
END;

```

```

IF {CHARACTERS WERE DISPLAYED} THEN
  {DISABLE SERIAL PORT INTERRUPT}
  {MOVE THE REMAINING CONTENTS OF THE FIFO UP TO THE BEGINNING OF THE FIFO}
  {ENABLE SERIAL PORT INTERRUPT}
  {SET THE VALID$RECEPTION FLAG}

```

```

IF {THE FIFO IS EMPTY} THEN {CLEAR THE "SERIAL$INT FLAG AND RETURN"}

```

```

IF {THE NEXT CHARACTER IS AN "ESC" CODE } THEN

```

```

DO:
  {LOOK AT THE CHARACTER IN THE FIFO AFTER THE ESC CODE AND CALL THE CORRECT SUBROUTINE}

```

```

  ;
  CALL UP$CURSER;          /* ESC A */
  CALL DOWN$CURSER;        /* ESC B */
  CALL RIGHT$CURSER;       /* ESC C */
  CALL LEFT$CURSER;        /* ESC D */
  CALL CLEAR$SCREEN;        /* ESC E */
  CALL MOV$CURSER;          /* ESC F */
  ;
  CALL HOME;               /* ESC H */
  ;
  CALL ERASE$FROM$CURSER$TO$END$OF$SCREEN; /* ESC J */
  CALL BLINK;              /* ESC K */

```

```

  {DISABLE THE SERIAL PORT INTERRUPT}
  {MOVE THE REMAINING CONTENTS OF THE FIFO UP TO THE BEGINNING OF THE FIFO}
  {ENABLE THE SERIAL PORT INTERRUPT}
  {SET THE "VALID$RECEPTION" FLAG}

```

```

IF {THE FIFO IS EMPTY} THEN {CLEAR THE SERIAL$INT FLAG AND RETURN}
END;

```

```

IF {THE NEXT CHARACTER IS A CONTROL CODE} THEN
DO;
  {CALL THE RIGHT SUBROUTINE}

  CALL LEFT$CURSER;          /* CTL H */
  CALL LINE$FEED;           /* CTL J */
  CALL CLEAR$SCREEN;        /* CTL L */
  CALL CARRIAGE$RETURN;     /* CTL M */

  {DISABLE THE SERIAL PORT INTERRUPT}
  {MOVE THE REMAINING CONTENTS OF THE FIFO UP TO THE BEGINNING OF THE FIFO}
  {ENABLE THE SERIAL PORT INTERRUPT}
  {SET THE "VALID$RECEPTION" FLAG}
END;

IF {NO VALID CODE WAS RECEIVED ("VALID$RECEPTION" IS 0)} THEN
  {THROW THE CHARACTER OUT AND MOVE THE REMAINING CONTENTS OF THE FIFO}
  {UP TO THE BEGINNING}

IF {THE FIFO IS EMPTY} THEN {CLEAR THE SERIAL$INT FLAG AND RETURN}

END DECIPHER;

/*
  PROCEDURE DISPLAY: THIS PROCEDURE WILL TAKE THE BYTE IN RAM LABELED
  RECEIVE AND PUT IT INTO THE DISPLAY RAM. */

DISPLAY:

  {PUT INTO THE DISPLAY RAM LOCATION POINTED TO BY "DISPLAY$RAM$POINTER"
  THE CONTENTS OF RECEIVE}

  IF {THE END OF THE DISPLAY MEMORY HAS BEEN REACHED} THEN
    {RESET "DISPLAY$RAM$POINTER" TO THE BEGINNING OF THE RAM}
  ELSE
    {INCREMENT "DISPLAY$RAM$POINTER"}

  IF {THE CURSER IS IN THE LAST COLUMN OF THE CRT DISPLAY} THEN
  DO;
    {MOVE THE CURSER BACK TO THE BEGINNING OF THE LINE}
    IF {THE NEW DISPLAY RAM LOCATION HAS A END-OF-LINE CHARACTER IN IT} THEN
      CALL FILL;

    IF {THE CURSER IS ON THE LAST LINE OF THE CRT DISPLAY} THEN
      CALL SCROLL;
    ELSE
      {MOVE THE CURSER TO THE NEXT LINE}
  END;
  ELSE
    {INCREMENT THE CURSER TO THE NEXT LOCATION}

  {TURN THE CURSER ON }
  CALL LOADCURSER;
END DISPLAY;

```



```

/*      PROCEDURE LINE$FEED      */
/*      THIS PROCEDURE MOVES THE CURSOR TO THE 0,0 POSITION.      */

LINE$FEED:
    IF {THE CURSOR IS IN THE LAST LINE OF THE CRT DISPLAY} THEN
        CALL SCROLL;
    ELSE
        DO;
            {MOVE THE CURSOR TO THE NEXT LINE}
            {TURN THE CURSOR ON}
            CALL LOAD$CURSER;
        END;
    IF {THE DISPLAY$RAM$POINTER IS ON THE LAST LINE IN THE DISPLAY RAM} THEN
        {MOVE THE DISPLAY$RAM$POINTER TO THE FIRST LINE IN THE DISPLAY RAM}
    ELSE
        {MOVE THE DISPLAY$RAM$POINTER TO THE NEXT LINE IN THE DISPLAY RAM}
    IF {THE FIRST CHARACTER IN THE NEW LINE CONTAINS AN END-OF-LINE CHARACTER } THEN
        CALL FILL;
    END LINE$FEED;

```

```

/*      PROCEDURE SCROLL      */
/*      THIS PROCEDURE IS USED IN CONNECTION WITH SCROLL$      */

SCROLL:
    CALL BLANK;
    {DISABLE VERTICAL RETRACE INTERRUPT}
    IF {THE FIRST LINE OF THE CRT CONTAINS THE LAST LINE OF THE DISPLAY MEMORY} THEN
        {MOVE THE POINTER "LINE0" TO THE BEGINNING OF THE DISPLAY MEMORY}
    ELSE
        {MOVE "LINE0" TO THE NEXT LINE IN THE DISPLAY MEMORY}
    {ENABLE VERTICAL RETRACE INTERRUPT}
    END SCROLL;

```

```

/*      PROCEDURE CLEAR SCREEN      */
/*      THIS PROCEDURE MOVES THE CURSOR TO THE 0,0 POSITION.      */

CLEAR$SCREEN:
    CALL HOME;
    CALL ERASE$FROM$CURSER$TO$END$OF$SCREEN;
    END CLEAR$SCREEN;

```

```
/* PROCEDURE HOME: THIS PROCEDURE MOVES THE CURSER TO THE 0,0 POSITION */
```

```
HOME:
```

```
{MOVE THE CURSER POSITION TO THE UPPER LEFT HAND CORNER OF THE CRT}
{TURN THE CURSER ON}
CALL LOAD$CURSER;
{MOVE THE DISPLAY$RAM$POINTER TO THE CORRECT LOCATION IN THE DISPLAY RAM}
```

```
END HOME;
```

```
/* PROCEDURE ERASE FROM CURSER TO END OF SCREEN: */
```

```
ERASE$FROM$CURSER$TO$END$OF$SCREEN:
```

```
CALL BLINE;
```

```
/* ERASE CURRENT LINE */
```

```
IF {THE CURSER IS NOT ON THE LAST LINE OF THE CRT DISPLAY} THEN
  STARTING WITH THE NEXT LINE, PUT AN END-OF-LINE CHARACTER (0F1H)
  IN THE DISPLAY RAM LOCATIONS THAT CORRESPOND TO THE BEGINNING OF
  THE CRT DISPLAY LINES UNTIL THE BOTTOM OF THE CRT SCREEN HAS BEEN REACHED}
END;
```

```
END ERASE$FROM$CURSER$TO$END$OF$SCREEN;
```

```
/*PROCEDURE MOV$CURSER: THIS PROCEDURE IS USED IN CONJUNCTION WITH WORDSTAR
  IF A ESC F IS RECEIVED FROM THE HOST COMPUTER, THE TERMINAL CONTROLLER WILL
  READ THE NEXT TWO BYTE TO DETERMINE WHERE TO MOVE THE CURSER. THE FIRST BYTE
  IS THE ROW INFORMATION FOLLOWED BY THE COLUMN INFORMATION */
```

```
MOV$CURSER:
```

```
{WAIT UNTIL THE FIFO HAS RECEIVED THE NEXT TWO CHARACTERS}
{MOVE THE CURSER TO THE LOCATION SPECIFIED IN THE ESCAPE SEQUENCE}
{MOVE THE DISPLAY$RAM$POINTER TO THE CORRECT LOCATION}
```

```
IF THE FIRST CHARACTER IN THE NEW LINE HAS AN END-OF-LINE CHARACTER} THEN
  CALL FILL;
END;
```

```
{DISABLE THE SERIAL PORT INTERRUPT}
{MOVE THE REMAIN CONTENTS OF THE FIFO UP TWO LOCATIONS IN MEMORY}
{DECREMENT THE FIFO BY TWO}
{ENABLE THE SERIAL PORT INTERRUPT}
```

```
END MOV$CURSER;
```

```
/* PROCEDURE LEFT CURSER: THIS PROCEDURE MOVES THE CURSER LEFT ONE COLUMN
  BY SUBTRACTING 1 OF THE CURSER COLUMN RAM LOCATION THEN CALL LOAD CURSER */
```

```
LEFT$CURSER:
```

```
IF {THE CURSER IS NOT IN THE FIRST LOCATION OF A LINE} THEN
```

```
DO;
```

```
{MOVE THE CURSER LEFT BY ONE LOCATION}
{TURN THE CURSER ON}
CALL LOAD$CURSER;
{DECREMENT THE DISPLAY$RAM$POINTER BY ONE}
```

```
END;
```

```
END LEFT$CURSER;
```

```

/*      PROCEDURE RIGHT CURSER: THIS PROCEDURE MOVES THE CURSER RIGHT ONE COLUMN
      BY ADDING 1 TO THE CURSER COLUMN RAM LOCATION THEN CALL LOAD CURSER */

```

```

RIGHT$CURSER:

```

```

IF {THE CURSER IS NOT IN THE LAST POSITION OF THE CRT LINE} THEN
DO;
    {MOVE THE CURSER RIGHT BY ONE LOCATION}
    {TURN THE CURSER ON}
    CALL LOAD$CURSER;
    {INCREMENT THE DISPLAY$RAM$POINTER BY ONE}
END;

```

```

END RIGHT$CURSER;

```

```

/*      PROCEDURE UP CURSER: THIS PROCEDURE MOVES THE CURSER UP ONE ROW
      BY SUBTRACTING 1 TO THE CURSER ROW RAM LOCATION THEN CALL LOAD CURSER */

```

```

UP$CURSER:

```

```

IF {THE CURSER IS NOT ON THE FIRST LINE OF THE CRT DISPLAY} THEN
DO;
    {MOVE THE CURSER UP ONE LINE}
    {TURN ON THE CURSER}
    CALL LOAD$CURSER;

```

```

IF {THE DISPLAY$RAM$POINTER IS IN THE FIRST LINE OF DISPLAY MEMORY} THEN
    {MOVE THE DISPLAY$RAM$POINTER TO THE LAST LINE OF DISPLAY MEMORY}
ELSE
    {MOVE THE DISPLAY$RAM$POINTER UP ONE LINE IN DISPLAY MEMORY}

```

```

IF {THE FIRST LOCATION OF THE NEW LINE CONTAINS AN END-OF-LINE CHARACTER} THEN
    CALL FILL;

```

```

END;

```

```

END UP$CURSER;

```

```

/*      PROCEDURE DOWN CURSER: THIS PROCEDURE MOVES THE CURSER DOWN ONE ROW
      BY ADDING 1 TO THE CURSER ROW RAM LOCATION THEN CALL LOAD CURSER */

```

```

DOWN$CURSER:

```

```

IF {THE CURSER IS NOT ON THE LAST LINE OF THE CRT DISPLAY} THEN
DO;
    {TURN THE CURSER ON}
    {MOVE THE CURSER TO THE NEXT LINE}
    CALL LOAD$CURSER;

```

```

IF {THE DISPLAY$RAM$POINTER IS NOT ON THE LAST LINE OF THE DISPLAY MEMORY} THEN
    {MOVE THE DISPLAY$RAM$POINTER TO THE NEXT LINE IN THE DISPLAY MEMORY}
ELSE
    {MOVE THE DISPLAY$RAM$POINTER TO THE FIRST LINE IN THE DISPLAY MEMORY}

```

```

IF {THE FIRST CHARACTER IN THE NEW LINE IS AN END-OF-LINE CHARACTER} THEN
    CALL FILL;

```

```

END;

```

```

END DOWN$CURSER;

```

```

/*      PROCEDURE CARRIAGE$RETURN
CARRIAGE$RETURN:
{MOVE THE DISPLAY$RAM$POINTER TO THE BEGINNING OF THE CURRENT LINE IN THE DISPLAY MEMORY}
{MOVE THE CURSER TO THE BEGINNING OF THE CURRENT LINE OF THE CRT DISPLAY}
{TURN THE CURSER ON}
CALL LOAD$CURSER;

END CARRIAGE$RETURN;

/*      PROCEDURE LOAD CURSER: LOAD CURSER TAKES THE VALUE HELD IN RAM AND
LOADS IT INTO THE 8276 CURSER REGISTER. */

LOAD$CURSER:
PROCEDURE;
IF {THE CURSER IS ON} THEN
    {MOVE THE CURSER BACK ONTO THE CRT DISPLAY}
{DISABLE BUFFER INTERRUPT}
{WRITE TO THE 8276 CURSER REGISTERS THE X,Y LOCATIONS}
{ENABLE BUFFER INTERRUPT}

END LOAD$CURSER;

/*      PROCEDURE CHECK BAUD RATE: THIS PROCEDURE READS THE THREE PORT PINS ON P1 AND SETS UP
THE SERIAL PORT FOR THE SPECIFIED BAUD RATE */

CHECK$BAUD$RATE:
{SET TIMER 1 TO MODE 1 AND AUTO RELOAD}
{TURN TIMER ON}
{ENABLE SERIAL PORT INTERRUPT}
{READ BAUD RATE SWITCHES AND SET UP RELOAD VALUE}

;
TH1=040H; /* 00 IS NOT ALLOWED */
TH1=0A0H; /* 150 BAUD */
TH1=0D0H; /* 300 BAUD */
TH1=0E8H; /* 600 BAUD */
TH1=0F4H; /* 1200 BAUD */
TH1=0FAH; /* 2400 BAUD */
TH1=0FDH; /* 4800 BAUD */
          /* 9600 BAUD */

END CHECK$BAUD$RATE;

```



```

/*      PROCEDURE READER: THIS PROCEDURE IS WRITTEN IN ASSEMBLY LANGUAGE. THE
EXTERNAL PROCEDURE SCANS THE 8 LINES OF THE KEYBOARD AND READS THE RETURN
LINES. THE STATUS OF THE 8 RETURN LINES ARE THEN STORED IN INTERNAL
MEMORY ARRAY CALLED CURRENT$KEY */

```

READER:

```

{INITIALIZE FLAGS "KEY0"=0, "SAME"=1, 0 COUNTER=0}

DO UNTIL {ALL 8 KEYBOARD SCAN LINES ARE READ}
  {READ KEYBOARD SCAN}
  IF {NO KEY WAS PRESSED} THEN
    {INCREMENT 0 COUNTER}
  ELSE
    IF {THE KEY PRESSED WAS NOT THE SAME KEY THAT WAS PRESSED THE LAST TIME
      THE KEYBOARD WAS READ} THEN
      {CLEAR "SAME" AND WRITE NEW SCAN RESULT TO CURRENT$KEY RAM ARRAY}
  END;

IF {ALL 8 SCANS DIDN'T HAVE A KEY PRESSED (0 COUNTER=8)} THEN
  {SET KEY0, AND CLEAR SAME}

```

END READER;

```

/*      PROCEDURE BLANK: THIS EXTERNAL PROCEDURE FILLS LINE0 WITH SPACES (20H ASCII)
DURING THE SCROLL ROUTINES.*/

```

BLANK:

```

DO I= {BEGINNING OF THE CRT DISPLAY (LINE0)} TO {LINE0 + 50H}
  {DISPLAY RAM POINTED TO BY "I" = SPACE (ASCII 20H)}
  NEXT I
END;

```

END BLANK;

```

/*      PROCEDURE BLINE: THIS EXTERNAL PROCEDURE BLANKS FROM THE CURSOR TO THE END OF
THE DISPLAY LINE */

```

BLINE:

```

DO I= {CURRENT CURSOR POSITION ON CRT DISPLAY} TO {END OF ROW}
  {DISPLAY RAM POINTED TO BY "I" = SPACE (ASCII 20H)}
  NEXT I
END;

```

END BLINE;

```

/*      PROCEDURE FILL: THIS EXTERNAL PROCEDURE FILLS A DISPLAY LINE WITH SPACES*/

```

FILL:

```

DO I= {BEGINNING OF THE LINE THAT THE CURSOR IS ON} TO {END OF THE ROW}
  {DISPLAY RAM POINTED TO BY "I" = SPACE (ASCII 20H)}
  NEXT I
END;

```

END FILL;

PL/M-51 COMPILER

REVISION 12-11-80

```

$EJECT
***** REGISTER TIB *****
1 1 CRT$CONTROLLER:
DO;

/***** DECLARE LITERALS *****/
2 1 DECLARE LLC LITERALLY 'LOCAL$LINE$CHANGE';
3 1 DECLARE REG LITERALLY 'REGISTER';
4 1 DECLARE CURRENT$KEY LITERALLY 'CURKEY';
5 1 DECLARE SERIAL$SERVICE LITERALLY 'SERBUF';
6 1 DECLARE DISPLAY$RAM$POINTER LITERALLY 'POINT';
7 1 DECLARE SERIAL$INT LITERALLY 'SERINT';
8 1 DECLARE TRANSMIT$INT LITERALLY 'TRNINT';
9 1 DECLARE CURSER$COLUMN LITERALLY 'CURSER';
10 1 DECLARE LAST$KEY LITERALLY 'LSTKEY';
11 1 DECLARE CURSER$COUNT LITERALLY 'COUNT';
12 1 DECLARE SCAN$INT LITERALLY 'SCAN';

/***** REGISTER DECLARATIONS FOR THE 8051 *****/
***** BYTE REGISTERS *****
13 1 DECLARE
P0 BYTE AT (80H) REG,
P1 BYTE AT (90H) REG,
P2 BYTE AT (0A0H) REG,
P3 BYTE AT (0B0H) REG,
PSW BYTE AT (0D0H) REG,
ACC BYTE AT (0E0H) REG,
B BYTE AT (0F0H) REG,
SP BYTE AT (81H) REG,
DPL BYTE AT (82H) REG,
DPH BYTE AT (83H) REG,
PCON BYTE AT (87H) REG,
TOCON BYTE AT (88H) REG,
TMOO BYTE AT (89H) REG,
TL0 BYTE AT (8AH) REG,
TL1 BYTE AT (8BH) REG,
TH0 BYTE AT (8CH) REG,
TH1 BYTE AT (8DH) REG,
IE BYTE AT (0A8H) REG,
IP BYTE AT (0B8H) REG,
SCON BYTE AT (98H) REG,
SBUF BYTE AT (99H) REG;

```

```

$EJECT
/***** BIT REGISTERS *****/

/***** PSW BITS *****/
14 1 DECLARE
    CY BIT AT (0D7H) REG,
    AC BIT AT (0D6H) REG,
    FO BIT AT (0D5H) REG,
    RS1 BIT AT (0D4H) REG,
    RS0 BIT AT (0D3H) REG,
    OV BIT AT (0D2H) REG,
    P BIT AT (0D0H) REG,

/***** TOON BITS *****/
    TF1 BIT AT (8FH) REG,
    TR1 BIT AT (8EH) REG,
    TF0 BIT AT (8DH) REG,
    TR0 BIT AT (8CH) REG,
    IE1 BIT AT (8BH) REG,
    IT1 BIT AT (8AH) REG,
    IE0 BIT AT (89H) REG,
    IT0 BIT AT (88H) REG,

/***** IE BITS *****/
    EA BIT AT (0AFH) REG,
    ES BIT AT (0ACH) REG,
    ET1 BIT AT (0ABH) REG,
    EX1 BIT AT (0AAH) REG,
    ET0 BIT AT (0A9H) REG,
    EX0 BIT AT (0A8H) REG,

/***** IP BITS *****/
    PS BIT AT (0BCH) REG,
    PT1 BIT AT (0BBH) REG,
    PX1 BIT AT (0BAH) REG,
    PT0 BIT AT (0B9H) REG,
    PX0 BIT AT (0B8H) REG,

/***** P3 BITS *****/
    RD BIT AT (0B7H) REG,
    WR BIT AT (0B6H) REG,
    T1 BIT AT (0B5H) REG,
    T0 BIT AT (0B4H) REG,
    INT1 BIT AT (0B3H) REG,
    INTO BIT AT (0B2H) REG,
    TXD BIT AT (0B1H) REG,
    RXD BIT AT (0B0H) REG,

/***** SOON BITS *****/
    SM0 BIT AT (9FH) REG,
    SM1 BIT AT (9EH) REG,
    SM2 BIT AT (9DH) REG,
    REN BIT AT (9CH) REG,
    TB8 BIT AT (9BH) REG,
    RB8 BIT AT (9AH) REG,
    TI BIT AT (99H) REG,
    RI BIT AT (98H) REG;

```


PL/M-51 COMPILER CRTCONTROLLER

PL/M-51 COMPILER CRTCONTROLLER

```

$EJECT
$IF SW1
/***** DECLARE CONSTANTS *****/

15 1 DECLARE LOW$SCAN(16) STRUCTURE
      (KEY(8) BYTE) CONSTANT

      ('890-',5CH,5EH,08H,00H,
/* SCAN 0, SHIFT KEY =0; 8,9,0,-,\,^, BACK SPACE */
      'uio',5EH,'@',0AH,7FH,
/* SCAN 1, SHIFT =0; u,i,o,p[,e, LINE FEED, DELETE */
      'jkl;:',00H,0DH,'7',
/* SCAN 2, SHIFT =0; j,k,l,;,;, RETURN, 7 */
      'm',2CH,'.',00H,'/',00H,00H,00H,
/* SCAN 3, SHIFT =0; m,COMMA,.., */
      00H,'azxcvbn',
/* SCAN 4, SHIFT =0; a,z,x,c,v,b,n */
      'y',00H,00H,' dfgh',
/* SCAN 5, SHIFT =0; y, SPACE, d,f,g,h */
      09H,'qwsert',00H,
/* SCAN 6, SHIFT =0; TAB,q,w,s,e,r,t */
      1BH,'123456',00H,
/* SCAN 7, SHIFT =0; ESC,1,2,3,4,5,6 */
      28H,29H,00H,'=' ,7CH,7EH,08H,00H,
/* SCAN 0, SHIFT =1; (,),=,|,~, BACK SPACE */
      'UIOP',00H,00H,0AH,7FH,
/* SCAN 1, SHIFT =1; U,I,O,P, LINE FEED, DELETE */
      'JKL+*',00H,0DH,27H,
/* SCAN 2, SHIFT =1; J,K,L,+,*, RETURN, ' */
      'M<>',00H,3FH,00H,00H,00H,
/* SCAN 3, SHIFT =1; M,<,>,? */
      00H,'AZXCVEN',
/* SCAN 4, SHIFT =1; A,Z,X,C,V,B,N */
      'Y',00H,00H,' DFGH',
/* SCAN 5, SHIFT =1; Y, SPACE, D,F,G,H */
      09H,'QWERT',00H,
/* SCAN 6, SHIFT =1; TAB, Q,W,S,E,R,T */
      1BH,'!"$%&',00H);
/* SCAN 7, SHIFT =1; ESC,!,",#,$,%,& */
$ENDIF

```


PL/M-51 COMPILER CRTCONTROLLER

REVISION 12-12-81

```

$EJECT

18 1 DECLARE (
    ASCIIKEY,
    TRANSMIT$COUNT,
    TEMP,
    SHIFT,
    CURSER$COL,
    CURSER$COLUMN,
    CURSER$ROW,
    CURSER$COUNT,
    FIFO,
    RECEIVE) BYTE PUBLIC;

19 1 $IF SW1
    DECLARE LAST$KEY (8) BYTE PUBLIC;
    $ENDIF

    $IF SW2
    DECLARE LAST$KEY (2) BYTE PUBLIC;
    $ENDIF

20 1 DECLARE SERIAL (16) BYTE PUBLIC;

21 1 DECLARE DISPLAY$RAM (7CFH) BYTE AT (1000H) AUXILIARY;

22 1 DECLARE
    PARAMETER$ADDRESS BYTE AT (0000H) AUXILIARY,
    COMMAND$ADDRESS BYTE AT (0001H) AUXILIARY;

23 1 DECLARE (
    DISPLAY$RAM$POINTER,
    RASTER,
    LINE0,
    L) WORD PUBLIC;

```

PL/M-51 COMPILER CRTCONTROLLER

RELOCATED SOURCE PL/M-51 COMPILER

\$EJECT

TABLE

```

/* PROCEDURE READER: THIS PROCEDURE IS WRITTEN IN ASSEMBLY LANGUAGE. THE
EXTERNAL PROCEDURE SCANS THE 8 LINES OF THE KEYBOARD AND READS THE RETURN
LINES. THE STATUS OF THE 8 RETURN LINES ARE THEN STORED IN INTERNAL
MEMORY ARRAY CALLED CURRENT$KEY. THE PROCEDURE CONTROLS 2 STATUS FLAGS;
KEY0 AND SAME. KEY0 IS SET IF ALL 8 SCANS READ NO KEY WAS PRESSED.
IF ALL 8 SCANS ARE THE SAME AS THE LAST READING OF THE KEYBOARD, THEN
SAME IS SET. */

```

```

24 2 READER: PROCEDURE EXTERNAL;
25 1 END READER;

```

```

/* PROCEDURE BLANK: THIS EXTERNAL PROCEDURE FILLS LINE0 SCAN WITH SPACES (20H ASCII)
DURING THE SCROLL ROUTINES.*/

```

```

26 2 BLANK: PROCEDURE EXTERNAL;
27 1 END BLANK;

```

```

/* PROCEDURE BLINE: THIS EXTERNAL PROCEDURE BLANKS FROM THE CURSER TO THE END OF
THE DISPLAY LINE */

```

```

28 2 BLINE: PROCEDURE EXTERNAL;
29 1 END BLINE;

```

```

/* PROCEDURE FILL: THIS EXTERNAL PROCEDURE FILLS THE CURSER LINE
WITH SPACES*/

```

```

30 1 FILL:
PROCEDURE EXTERNAL;
31 1 END FILL;

```



```

/* PROCEDURE LOAD CURSER: LOAD CURSER TAKES THE VALUE HELD IN RAM AND
LOADS IT INTO THE 8276 CURSER REGISTERS. */

```

/* PROCEDURE CARRIAGES\$RETURN */

```

58 1 CARRIAGE$RETURN:
    PROCEDURE;
59 2 DISPLAY $RAM$POINTER=DISPLAY $RAM$POINTER-CURSER$COLUMN;
60 2 CURSER$COLUMN=0;
61 2 CURSER$ON=1;
62 2 CALL LOAD$CURSER;
63 1 END CARRIAGE$RETURN;

```

PL/M-51 COMPILER CRTCONTROLLER

REVISION 12-8-81

SEJECT

TABLE

/* PROCEDURE DOWN CURSER: THIS PROCEDURE MOVES THE CURSER DOWN ONE ROW
BY ADDING 1 TO THE CURSER ROW RAM LOCATION THEN CALL LOAD CURSER */

```

64 1  DOWN$CURSER:                                :STANQUIN: CHIO 1 31
        PROCEDURE;                                (STANQUIN)
65 2  IF CURSER$ROW < 18H THEN                    (STANQUIN)
66 3  DO;                                           (STANQUIN)
67 3      CURSER$ON=1;                             (STANQUIN)
68 3      CURSER$ROW=CURSER$ROW + 1;              (STANQUIN)
69 3      CALL LOAD$CURSER;                        (STANQUIN)
70 3      IF DISPLAY$RAM$POINTER < 780H THEN      (STANQUIN)
71 3          DISPLAY$RAM$POINTER=DISPLAY$RAM$POINTER + 50H; (STANQUIN)
72 3      ELSE
          DISPLAY$RAM$POINTER=(DISPLAY$RAM$POINTER-780H); (STANQUIN)
73 3      L=DISPLAY$RAM$POINTER-CURSER$COLUMN;    (STANQUIN)
74 3      IF DISPLAY$RAM(L)=0F1H THEN              /* LOOK FOR END OF*/
75 4      DO;                                     /* LINE CHARACTER */
76 4          CALL FILL;                          /* IF TRUE FILL LINE*/
77 4          DISPLAY$RAM(L)=20H;                 /* WITH SPACES */
78 4      END;
79 3  END;
80 1  END DOWN$CURSER;

```

/* PROCEDURE UP CURSER: THIS PROCEDURE MOVES THE CURSER UP ONE ROW
BY SUBTRACTING 1 TO THE CURSER ROW RAM LOCATION THEN CALL LOAD CURSER */

```

81 1  UP$CURSER:                                :STANQUIN: CHIO 1 31
        PROCEDURE;                                (STANQUIN)
82 2  IF CURSER$ROW > 0 THEN                    (STANQUIN)
83 3  DO;                                           (STANQUIN)
84 3      CURSER$ROW=CURSER$ROW - 1;              (STANQUIN)
85 3      CURSER$ON=1;                             (STANQUIN)
86 3      CALL LOAD$CURSER;                        (STANQUIN)
87 3      IF DISPLAY$RAM$POINTER < 50H THEN      (STANQUIN)
88 3          DISPLAY$RAM$POINTER=DISPLAY$RAM$POINTER+ 780H; (STANQUIN)
89 3      ELSE
          DISPLAY$RAM$POINTER=DISPLAY$RAM$POINTER - 50H; (STANQUIN)
90 3      L=DISPLAY$RAM$POINTER-CURSER$COLUMN;    (STANQUIN)
91 3      IF DISPLAY$RAM(L)=0F1H THEN              /* LOOK FOR END OF LINE*/
92 4      DO;                                     /* CHARACTER */
93 4          CALL FILL;                          /* IF TRUE FILL WITH */
94 4          DISPLAY$RAM(L)=20H;                 /* SPACES */
95 4      END;
96 3  END;
97 1  END UP$CURSER;

```

PL/M-51 COMPILER CRTCONTROLLER

REVISION 12-8-81

SELECT

```

/* PROCEDURE RIGHT CURSER: THIS PROCEDURE MOVES THE CURSER RIGHT ONE COLUMN
BY ADDING 1 TO THE CURSER COLUMN RAM LOCATION THEN CALL LOAD CURSER */

```

```

98 1  RIGHT$CURSER:
    PROCEDURE;
99 2  IF CURSER$COLUMN < 4FH THEN
100 3  DO;
101 3      CURSER$COLUMN=CURSER$COLUMN + 1;
102 3      CURSER$ON=1;
103 3      CALL LOAD$CURSER;
104 3      DISPLAY$RAM$POINTER=DISPLAY$RAM$POINTER +1;
105 3  END;
106 1  END RIGHT$CURSER;

```

```

/* PROCEDURE LEFT CURSER: THIS PROCEDURE MOVES THE CURSER LEFT ONE COLUMN
BY SUBTRACTING 1 TO THE CURSER COLUMN RAM LOCATION THEN CALL LOAD CURSER */

```

```

107 1  LEFT$CURSER:
    PROCEDURE;
108 2  IF CURSER$COLUMN > 0 THEN
109 3  DO;
110 3      CURSER$COLUMN=CURSER$COLUMN - 1;
111 3      CURSER$ON=1;
112 3      CALL LOAD$CURSER;
113 3      DISPLAY$RAM$POINTER=DISPLAY$RAM$POINTER -1;
114 3  END;
115 1  END LEFT$CURSER;

```

PL/M-51 COMPILER CRITCONTROLLER

BELLICHOUD BELLICHOUD 12-14-81

\$EJECT

/* PROCEDURE MOV\$CURSER: THIS PROCEDURE IS USED IN CONJUNCTION WITH WORDSTAR
IF A ESC F IS RECEIVED FROM THE HOST COMPUTER, THE TERMINAL CONTROLLER WILL
READ THE NEXT TWO BYTE TO DETERMINE WHERE TO MOVE THE CURSER. THE FIRST BYTE
IS THE ROW INFORMATION FOLLOWED BY THE COLUMN INFORMATION */

```

116 1  MOV$CURSER:
117 3  PROCEDURE;
118 3  DO WHILE FIFO<4;          /* WAIT UNTILL THE MOV$CURSER PARAMETERS*/
119 3  END;                      /* ARE RECEIVED INTO THE FIFO */
120 2  TEMP=CURSER$ROW;
121 2  CURSER$ROW=SERIAL(2);
122 3  IF CURSER$ROW>TEMP THEN
123 3  DO;
124 3  L=DISPLAY$RAM$POINTER+ ((CURSER$ROW-TEMP)*50H);
125 3  IF L>7CFH THEN          /* IF OUT OF RAM RANGE */
126 3  DISPLAY$RAM$POINTER=L-7D0H; /* RAP AROUND TO BEGINNING */
127 3  ELSE                      /* OF RAM */
128 2  DISPLAY$RAM$POINTER=L;
129 3  END;
130 2  ELSE
131 2  DO;
132 2  IF CURSER$ROW<TEMP THEN
133 2  DO;
134 2  L=(TEMP-CURSER$ROW)*50H;
135 2  IF DISPLAY$RAM$POINTER<L THEN          /* IF OUT OF RAM RANGE*/
136 2  DISPLAY$RAM$POINTER=(7D0H-(L-DISPLAY$RAM$POINTER)); /* RAP AROUND TO END OF RAM*/
137 2  ELSE
138 2  DISPLAY$RAM$POINTER=DISPLAY$RAM$POINTER-L;
139 2  END;
140 2  END;
141 2  TEMP=CURSER$COLUMN;
142 2  CURSER$COLUMN=SERIAL(3);
143 2  IF CURSER$COLUMN>TEMP THEN
144 2  DISPLAY$RAM$POINTER=DISPLAY$RAM$POINTER+ (CURSER$COLUMN-TEMP);
145 2  ELSE
146 2  DISPLAY$RAM$POINTER=DISPLAY$RAM$POINTER- (TEMP-CURSER$COLUMN);
147 2  CURSER$ON=1;
148 2  CALL LOAD$CURSER;
149 2  L=DISPLAY$RAM$POINTER-CURSER$COLUMN;
150 2  IF DISPLAY$RAM(L)=0F1H THEN          /* LOOK FOR END FO LINE CHARACTER*/
151 2  DO;
152 2  CALL FILL;
153 2  DISPLAY$RAM(L)=20H;          /* IF TRUE FILL WITH SPACES */
154 2  END;
155 2  ES=0;
156 1  DO I=2 TO FIFO-2;
157 1  SERIAL(I)=SERIAL(I+2);
158 1  END;
159 1  FIFO=FIFO-2;
160 1  ES=ENSP;
161 1  END MOV$CURSER;

```


PL/M-51 COMPILER CRT/CONTROLLER

REVISION 12-11-78

```

$EJECT
/* PROCEDURE ERASE FROM CURSER TO END OF SCREEN: */
157 1 ERASE$FROM$CURSER$TO$END$OF$SCREEN:
158 2 PROCEDURE;
159 2 CALL BLINE; /* ERASE CURRENT LINE */
160 3 IF CURSER$ROW < 18H THEN
161 3 DO;
162 3 L=DISPLAY$RAM$POINTER-CURSER$COLUMN+50H; /* GET NEXT LINE */
163 3 DO WHILE (L < 7D0H) AND (L <> (LINE0 AND 7FFH));
164 3 DISPLAY$RAM(L)=0F1H; /* ERASE UNTIL LINE0 OR */
165 3 L=L+50H; /* END OF DISPLAY RAM */
166 3 END;
167 3 L=0;
168 3 DO WHILE L <> (LINE0 AND 7FFH); /* ERASE UNTIL LINE0 */
169 3 DISPLAY$RAM(L)=0F1H;
170 3 L=L+50H;
171 3 END;
172 1 END ERASE$FROM$CURSER$TO$END$OF$SCREEN;

/* PROCEDURE HOME: THIS PROCEDURE MOVES THE CURSER TO THE 0,0 POSITION */
173 1 HOME:
174 2 PROCEDURE;
175 2 CURSER$ROW=00;
176 2 CURSER$COLUMN=00;
177 2 CURSER$ON=1;
178 2 CALL LOAD$CURSER;
179 1 DISPLAY$RAM$POINTER=(LINE0 AND 7FFH);
END HOME;

```

SELECT

/* PROCEDURE CLEAR SCREEN */

```
180 1 CLEAR$SCREEN:
      PROCEDURE;
181 2 CALL HOME;
182 2 CALL ERASE$FROM$CURSER$TO$END$OF$SCREEN;
183 1 END CLEAR$SCREEN;
```

/* PROCEDURE SCROLL */

```
184 1 SCROLL:
      PROCEDURE;
185 2 CALL BLANK;
186 2 EX0=0; /* DISABLE VERTICAL REFRESH INTERRUPT */
187 2 IF LINE0= 1F80H THEN
188 2     LINE0= 1800H;
189 2 ELSE
190 2     LINE0= LINE0+50H;
191 2 EX0=1; /* ENABLE VERTICAL REFRESH INTERRUPT */
      END SCROLL;
```

/* PROCEDURE LINE\$FEED */

```
192 1 LINE$FEED:
      PROCEDURE;
193 2 IF CURSER$ROW=18H THEN
194 2     CALL SCROLL;
195 2 ELSE
      DO;
196 3     CURSER$ROW= CURSER$ROW+1;
197 3     CURSER$CN=1;
198 3     CALL LOAD$CURSER;
199 3 END;
200 2 IF DISPLAY$RAM$POINTER > 77FH THEN
201 2     DISPLAY$RAM$POINTER=DISPLAY$RAM$POINTER-780H;
202 2 ELSE
203 2     DISPLAY$RAM$POINTER=DISPLAY$RAM$POINTER+50H;
204 2 L=DISPLAY$RAM$POINTER-CURSER$COLUMN;
205 2 IF DISPLAY$RAM(L)=0FH THEN /* LOOK FOR END OF LINE CHARACTER*/
206 2     DO;
207 3     CALL FILL; /* IF TRUE FILL WITH SPACES */
208 3     DISPLAY$RAM(L)=20H;
209 2 END;
      END LINE$FEED;
```

PL/M-51 COMPILER CRTCONTROLLER

PL/M-51 COMPILER CRTCONTROLLER

\$EJECT

PAGE 2

/* PROCEDURE DISPLAY: THIS PROCEDURE WILL TAKE THE BYTE IN RAM LABELED
RECEIVE AND PUT IT INTO THE DISPLAY RAM. */

```

210 1  DISPLAY:
211 2  PROCEDURE;
212 2  DISPLAY$RAM(DISPLAY$RAM$POINTER)=RECEIVE;
213 2  IF DISPLAY$RAM$POINTER=7CFH THEN /* IF END OF RAM */
214 2  DISPLAY$RAM$POINTER=000H; /* RAP AROUND TO BEGINNING */
215 2  ELSE
216 2  DISPLAY$RAM$POINTER=DISPLAY$RAM$POINTER+1;
217 2  IF CURSER$COLUMN=4FH THEN
218 2  DO;
219 2  CURSER$COLUMN=00H;
220 2  L=DISPLAY$RAM$POINTER;
221 2  IF DISPLAY$RAM(L)=0F1H THEN
222 2  DO;
223 2  CALL FILL;
224 2  DISPLAY$RAM(L)=20H;
225 2  END;
226 2  IF CURSER$ROW=18H THEN
227 2  CALL SCROLL;
228 2  ELSE
229 2  CURSER$ROW=CURSER$ROW+1;
230 2  END;
231 1  CURSER$COLUMN=CURSER$COLUMN+1;
232 2  CURSER$ON=1;
233 2  CALL LOADCURSER;
234 1  END DISPLAY;

```

/* DISPLAY SERIAL INTERRUPT WITH MOVING ECHO */

TO CASE R1

CALL PROCEDURE;

CALL DOWNCURSER;

CALL RIGHTCURSER;

CALL LEFTCURSER;

CALL CLEARCURSER;

CALL MOVECURSER;

CALL HOME;

CALL RAISEPTMIDCOUNTERS;

CALL ECHO;

PL/M-51 COMPILER CRTCONTROLLER

SEJECT

```

/*  PROCEDURE DECIPHER: THIS PROCEDURE DECODES THE HOST COMPUTER'S MESSAGES AND DETERMINES
    WHETHER IT IS A DISPLAYABLE CHARACTER, CONTROL SEQUENCE, OR AN ESCAPE SEQUENCE
    THE PROCEDURE THEN ACTS ACCORDINGLY */

```

```

232 1 DECIPHER:
233 2 PROCEDURE;
234 2 START$DECIPHER: * * * * *
235 3 VALID$RECEPTION=0;
236 3 I=0;
237 3 DO WHILE (I<FIFO) AND (SERIAL(I)>1FH) AND (SERIAL(I)<7FH);
238 3 RECEIVE=SERIAL(I);
239 3 CALL DISPLAY;
240 3 I=I+1;
241 3 END;
242 3 IF I>0 THEN
243 3 DO;
244 3 ES=0; /* DISABLE SERIAL INTERRUPT WHILE MOVING FIFO */
245 3 K=FIFO-I;
246 3 DO J=0 TO K; /* MOVE FIFO */
247 3 SERIAL(J)=SERIAL(I);
248 3 I=I+1;
249 3 END;
250 3 FIFO=K;
251 3 ES=ENSP; /* ENABLE SERIAL INTERRUPT */
252 3 VALID$RECEPTION=1;
253 3 END;
254 3 IF FIFO=0 THEN
255 3 DO;
256 3 SERIAL$INT=0;
257 3 GOTO END$DECIPHER;
258 3 END;
259 3 IF (SERIAL(0)=1BH) THEN
260 3 DO;
261 3 IF (ESC$SEQ=1) AND (FIFO<2) THEN
262 3 GOTO END$DECIPHER;
263 3 K=(SERIAL(1) AND 5FH)-40H;
264 3 IF (K >00H) AND (K<0CH) THEN
265 3 DO;
266 3 DO CASE K;
267 3 ;
268 3 CALL UP$CURSER; /* ESC A */
269 3 CALL DOWN$CURSER; /* ESC B */
270 3 CALL RIGHT$CURSER; /* ESC C */
271 3 CALL LEFT$CURSER; /* ESC D */
272 3 CALL CLEAR$SCREEN; /* ESC E */
273 3 CALL MOV$CURSER; /* ESC F */
274 3 ;
275 3 CALL HOME; /* ESC H */
276 3 ;
277 3 CALL ERASE$FROM$CURSER$TO$END$OF$SCREEN; /* ESC J */
278 3 CALL BLINK; /* ESC K */
279 3 END;
280 3 END;
281 3 ES=0; /* DISABLE SERIAL INTERRUPTS WHILE MOVING FIFO

```


PL/M-51 COMPILER CRTCONTROLLER

RECEIVED 12-1-77

```

280 4      DO I=0 TO (FIFO-2);
281 4          SERIAL(I)=SERIAL(I+2);      /* MOVE FIFO */
282 4      END;
283 3      FIFO=FIFO-2;
284 3      ES=ENSP;                          /* ENABLE SERIAL INTERRUPTS */
285 3      VALID$RECEPTION=1;
286 3      IF FIFO=0 THEN
287 4          DO;
288 4              SERIAL$INT=0;
289 4              GOTO END$DECIPHER;
290 4          END;
291 3      END;
292 2      IF (SERIAL(0)> 07H) AND (SERIAL(0)<0EH) THEN
293 3      DO;
294 4          DO CASE (SERIAL(0) -08H);
295 4              CALL LEFT$CURSER;          /* CTL H */
296 4              ;
297 4              CALL LINE$FEED;           /* CTL J */
298 4              ;
299 4              CALL CLEAR$SCREEN;         /* CTL L */
300 4              CALL CARRIAGE$RETURN;     /* CTL M */
301 4          END;
302 3          ES=0;                          /* DISABLE SERIAL INTERRUPTS WHILE MOVING FIFO */
303 4          DO I=0 TO (FIFO-1);
304 4              SERIAL(I)=SERIAL(I+1);    /* MOVE FIFO */
305 4          END;
306 3          FIFO=FIFO-1;
307 3          ES=ENSP;                          /* ENABLE SERIAL INTERRUPTS */
308 3          VALID$RECEPTION=1;
309 3      END;
310 2      IF VALID$RECEPTION=0 THEN
311 3      DO;
312 3          ES=0;
313 4          DO I=0 TO (FIFO-1);          /* IF CHARACTER IS UNRECOGNIZED THEN */
314 4              SERIAL(I)=SERIAL(I+1);    /* TRASH IT */
315 4          END;
316 3          FIFO=FIFO-1;
317 3          ES=ENSP;
318 3      END;
319 2      IF FIFO=0 THEN
320 2          SERIAL$INT=0;
321 2      END$DECIPHER;
321 2      END DECIPHER;

```

PL/M-51 COMPILER CRT/CONTROLLER

CONTROLLER PL/M-51 COMPILER

```

$EJECT
/* PROCEDURE TRANSMIT- THIS PROCEDURE LOOKS AT THE CLEAR TO SEND PIN FOR AN ACTIVE
LOW SIGNAL. ONCE THE MAIN COMPUTER SIGNALS THE 8051 THE ASCII CHARACTER IS PUT
INTO THE SERIAL PORT.*/

322 1  TRANSMIT:
      PROCEDURE;
323 2  IF LOCAL$LINE =1 THEN
324 3  DO;
325 4      DO WHILE (CLEAR$TO$SEND=1) OR (TRANSMIT$INT=0);
326 4      END;
327 3      SBUF=ASCIIKEY;
328 3      TRANSMIT$INT=0;
329 3  END;
330 2  ELSE
      DO;
331 3      SERIAL(FIFO)=ASCIIKEY;
332 3      FIFO=FIFO+1;
333 3      SERIAL$INT=1;
334 3  END;
335 1  END TRANSMIT;

/* PROCEDURE AUTO$REPEAT; THIS PROCEDURE WILL PERFORM AN AUTO REPEAT FUNCTION
AFTER A FIXED DELAY PERIOD */

336 1  AUTO$REPEAT:
      PROCEDURE;
337 2  IF NEWSKEY=1 THEN
338 3  DO;
339 3      TRANSMIT$TOGGLE=0;
340 3      TRANSMIT$COUNT=00H;
341 3      CALL TRANSMIT;          /* FIRST CHARACTER */
342 3      NEWSKEY=0;
343 3  END;
344 2  ELSE
      DO;
345 3      IF TRANSMIT$COUNT <> 00H THEN
346 4      DO;
347 4          TRANSMIT$COUNT=TRANSMIT$COUNT+1;
348 4          IF TRANSMIT$COUNT=0FFH THEN /*DELAY BETWEEN FIRST CHARACTER AND THE SECOND */
349 5          DO;
350 5              CALL TRANSMIT;          /*SECOND CHARACTER */
351 5              TRANSMIT$COUNT=00;
352 5          END;
353 4      END;
354 3      ELSE
          DO;
355 4          CURSER$ON=1;
356 4          CURSER$COUNT=0;
357 4          IF TRANSMIT$TOGGLE = 1 THEN /* 2 VERT FRAMES BETWEEN 3RD TO NTH CHARACTER */
358 4          CALL TRANSMIT;          /* 3RD THROUGH NTH CHARACTER */
359 4          TRANSMIT$TOGGLE= NOT TRANSMIT$TOGGLE;
360 4      END;
361 3  END;
362 1  END AUTO$REPEAT;

```

PL/M-51 COMPILER CRTCONTROLLER

```

SEJECT
/***** START MAIN PROGRAM *****/
/* BEGIN BY PUTTING ASCII CODE FOR BLANK IN THE DISPLAY RAM; */

363 1 INIT:
364 2 DO L=0 TO 7FH;
365 2 DISPLAY$RAM(L)=20H;
END;

/* INITIALIZE POINTERS, RAM BITS, ETC. */

366 1 ESC$SEQ=0;
367 1 SCAN$INT=0;
368 1 SERIAL$INT=0;
369 1 FIFO=0;
370 1 CURSER$COUNT=0;
371 1 LLC=0;
372 1 DATA$TERMINAL$READY=1;
373 1 TC0N=05H;
374 1 LINE0=1800H;
375 1 RASTER=1800H;
376 1 DISPLAY$RAM$POINTER=0000H;
377 1 TRANSMIT$INT=1;

$IF SW1
DO I=0 TO 7;
LAST$KEY(I)=00H;
END;

381 1 VALID$KEY=0;
382 1 LAST$SHIFT$KEY=1;
383 1 LAST$CONTROL$KEY=1;
384 1 LAST$CAP$LOCK=1;
SENDIF

$IF SW2
RCV$FLG=0;
SYNC=0;
BY$FIN=0;
KBD$INT=0;
ERROR=0;
SENDIF

/* INITIALIZE THE 8276 */

385 1 COMMAND$ADDRESS=00H;
386 1 PARAMETER$ADDRESS=4FH;
387 1 PARAMETER$ADDRESS=58H;
388 1 PARAMETER$ADDRESS=89H;
389 1 PARAMETER$ADDRESS=0F9H;

/* RESET THE 8276 */
/* NORMAL ROWS, 80 CHARACTER/ROW */
/* 2 ROW COUNTS PER VERTICAL RETRACE */
/* 25 ROWS PER FRAME */
/* LINE 9 IS THE UNDERLINE POSITION */
/* 10 LINES PER ROW */
/* OFFSET LINE COUNTER, NON-TRANSPARENT FIELD ATTRIBUTE

```

PL/M-51 COMPILER CRIOCONTROLLER

REVISION 1.0

NON-BLINKING UNDERLINE CURSER, 20 CHARACTER COUNTS PER
HORIZONTAL RETRACE */

```

390 1  TEMP=COMMAND$ADDRESS;
391 1  CURSER$COLUMN=00H;
392 1  CURSER$ROW=00H;
393 1  CURSER$COL=00H;
394 1  CALL LOAD$CURSER;
395 1  TEMP=COMMAND$ADDRESS;

396 1  COMMAND$ADDRESS=0E0H;          /* PRESET 8276 COUNTERS */
397 1  TEMP=COMMAND$ADDRESS;

398 1  COMMAND$ADDRESS=23H;          /* START DISPLAY */
399 1  COMMAND$ADDRESS=0A0H;          /* ENABLE INTERRUPTS */
400 1  TEMP=COMMAND$ADDRESS;

      /* SET UP INTERRUPTS AND PRIORITIES */

      $IF SW1
401 1  IP=10H;                        /* SERIAL PORT HAS HIGHEST PRIORITY */
402 1  IE=85H;                        /* ENABLE 8051 EXTERNAL INTERRUPTS */
      $ENDIF

      $IF SW2
      IP=10H;                        /* SERIAL PORT HAS HIGHEST PRIORITY */
      IE=87H;                        /* ENABLE TIMER0 INTERRUPT */
      TMOD=05H;                      /* TIMER 0 =EVENT COUNTER */
      TL0=0FFH;
      TH0=0FFH;                      /* INITIALIZE COUNTER TO FFFFH */
      TR0=1;
      $ENDIF

      /* PROCEDURE SCANNER: THIS PROCEDURE SCANS THE KEYBOARD AND DETERMINES IF A
      SINGLE VALID KEY HAS BEEN PUSHED. IF TRUE THEN THE ASCII EQUIVALENT
      WILL BE TRANSMITTED TO THE HOST COMPUTER.*/

403 1  SCANNER:
404 1  EA=1;
405 1  DATA$TERMINAL$READY=0;
406 2  IF CURSER$COUNT=1FH THEN      /* PROGRAMMABLE CURSER BLINK */
407 2  DO;
408 2  CURSER$ON=NOT CURSER$ON;
409 2  CURSER$COUNT=00;
410 2  IF CURSER$ON=0 THEN
411 2  CURSER$COL=7FH;
412 2  CALL LOAD$CURSER;
413 1  END;
414 1  IF LLC<>LOCAL$LINE THEN        /* IF LOCAL/LINE HAS CHANGED STATUS */
415 2  DO;
416 3  IF LOCAL$LINE=0 THEN
417 3  ENSP=0;
418 3  ES=0;
419 3  END;
420 2  ELSE
      CALL CHECK$BAUD$RATE;
      LLC=LOCAL$LINE;
421 2  END;
422 2  $IF SW1
423 2  DO WHILE SCAN$INT=0;           /* WAIT UNTIL VERTICAL RETRACE BEFORE */
424 2  IF SERIAL$INT=1 THEN          /* SCANNING THE KEYBOARD */
425 2  CALL DECIPHER;
426 2  END;

```


PL/M-51 COMPILER CRTCONTROLLER

BELLIGHTWOOD CRITICAL 22-5-1

```

$EJECT
427 1 CALL READER;
428 1 IF VALID$KEY =1 AND SAME=1 AND (LAST$SHIFT$KEY=SHIFT$KEY) AND (LAST$CAP$LOCK=CAP$LOCK) AND (LAST$CONTROL$KEY=CONTROL$KEY) THEN
429 1 CALL AUTO$REPEAT;
430 1 ELSE
DO;
431 2 IF KEY0=0 AND SAME=0 THEN
432 3 DO;
433 3 TEMP =0;
434 3 K=0;
435 4 DO WHILE LAST$KEY (K)=0;
436 4 K=K+1;
437 4 END;
438 3 TEMP=LAST$KEY (K);
439 4 DO I=(K+1) TO 7;
440 4 TEMP=TEMP+LAST$KEY (I);
441 4 END;
442 3 IF TEMP=LAST$KEY (K) THEN
443 4 DO;
444 4 J=0;
445 5 DO WHILE (TEMP AND 01H)=0;
446 5 TEMP=SHR (TEMP,1);
447 5 J=J+1;
448 5 END;
449 4 IF TEMP >1 THEN
450 5 DO;
451 5 VALID$KEY=0;
452 5 NEWSKEY=0;
453 5 END;
454 4 ELSE
DO;
455 5 IF CONTROL$KEY=0 THEN
456 5 ASCII$KEY=(LOW$SCAN (K).KEY (J)) AND 1FH;
457 5 ELSE
DO;
458 6 IF SHIFT$KEY=0 THEN
459 6 ASCII$KEY=LOW$SCAN (K+08H).KEY (J);
460 6 ELSE
DO;
461 7 ASCII$KEY=LOW$SCAN (K).KEY (J);
462 7 IF (CAP$LOCK=0) AND (ASCII$KEY>60H) AND (ASCII$KEY<7BH) THEN
463 7 ASCII$KEY=ASCII$KEY-20H;
464 7 IF LLC=0 THEN
DO;
465 8 IF ASCII$KEY=1BH THEN
466 8 ESC$SEQ=1;
467 8 ELSE
468 8 ESC$SEQ=0;
469 8 END;
470 7 END;
471 6 END;
472 5 LAST$SHIFT$KEY=SHIFT$KEY;
473 5 LAST$CAP$LOCK=CAP$LOCK;
474 5 LAST$CONTROL$KEY=CONTROL$KEY;
475 5 VALID$KEY=1;
476 5 NEWSKEY=1;
477 5 END;
478 4 END;
479 3 ELSE
DO;
480 4 VALID$KEY=0;
481 4 NEWSKEY=0;
482 4 END;
483 3 END;
484 2 END;

$ENDIF

```

\$EJECT

\$IF SW2

IF SERIAL\$INT=1 THEN

CALL DECIPHER;

IF KBDINT =1 THEN

DO;

IF ERROR =0 THEN

DO;

ASCII\$KEY=LST\$KEY(1);

NEWSKEY=1;

CALL AUTO\$REPEAT;

KBDINT=0;

END;

ERROR=0;

KBDINT=0;

END;

SENDIF

485 1 GOTO SCANNER;

486 1 END;

MODULE INFORMATION:

CODE SIZE

CONSTANT SIZE

DIRECT VARIABLE SIZE

INDIRECT VARIABLE SIZE

BIT SIZE

BIT-ADDRESSABLE SIZE

AUXILIARY VARIABLE SIZE

MAXIMUM STACK SIZE

REGISTER-BANK(S) USED:

1056 LINES READ

0 PROGRAM ERROR(S)

END OF PL/M-51 COMPILATION

(STATIC+OVERLAYABLE)

= 08E6H 2278D

= 0080H 128D

= 2DH+00H 45D+ 0D

= 00H+00H 0D+ 0D

= 10H+00H 16D+ 0D

= 00H+00H 0D+ 0D

= 0000H 0D

= 0000CH 12D

0

MCS-51 MACRO ASSEMBLER CNTASM

MCS-51 MACRO ASSEMBLER CNTASM

ISIS-II MCS-51 MACRO ASSEMBLER V2.1
 OBJECT MODULE PLACED IN :F1:CNTASM.OBJ
 ASSEMBLER INVOKED BY: ASM51 :F1:CR[ASM.SRC

```

1  ;
2  ;
3  ;
4  ;
5  ;
6  ;
7  PUBLIC BLANK
8  PUBLIC BLINE
9  PUBLIC FILL
10 EXTRN DATA (LINE0,MASTER,POINT,SERIAL,FIFO,CURSER,COUNT,L)
11 EXTRN BIT (SERINT,ESCSEQ,THNINT,SCAN)
12 ;
13 ;
14 ;CSEG AT(03H)
15 SJMP VENT ;RESET RASTER TO LINE0 AND SCAN KEYBUAI
16 ;
17 ; EXTRN CODE (DEIACH)
18 ;CSEG AT(0BH)
19 ; LJMP DEIACH ;NEEDED IF DECODED KEYBOARD IS USED
20 ;
21 ;CSEG AT(013H)
22 SJMP BUFFER ;FILL 8276 ROW BUFFER
23 ;
24 ;CSEG AT(023H)
25 SJMP SERBUF ;STICK SERIAL INFORMATION INTO THE FIF
26 ;
27 ;CSEG
28 ;
29 VENT: PUSH PSW ;PUSH REG USED BY PLM51
30 PUSH ACC
31 PUSH 00H
32 MOV MASTER,LINE0 ;REINITIALIZE MASTER TO LINE0
33 MOV MASTER+1,LINE0+1
34 MOV NO,#01H ;CLR 8276 INTERRUPT FLAG
35 MOVX A,0RU
36 INC COUNT ;INCK CURSER COUNT REGISTER
37 SETB SCAN ;FOR DEBOUNCE ROUTINE
38 POP 00H ;POP REGISTERS
39 POP ACL
40 POP PSW
41 RETI
42 ;
43 ;
44 BUFFER: PUSH PSW ;PUSH ALL REG USED BY PLM51 CODE
45 PUSH ACL
46 PUSH DPL
47 PUSH UPL
48 ACALL DVA ;FILL 8276 ROW BUFFER
49 POP UPL
50 POP DPL
51 POP ACL
52 POP PSW
53 RETI
54 ;
55 +1 SEJELT

```

MCS-51 MACRO ASSEMBLER CNTASM

HEATH 8085-51 MACRO ASSEMBLER

```

LUC OBJ      LINE SOURCE
0052 309904   56  SERBUF: JNB 099H,CVER ;IF TRANSMIT BIT NOT SET THEN CHECK RECEIVE
0055 0299     58  CLN 099H ;CLR TRANSMISSION INTERRUPT FLAG
0057 0200 F    59  SETB TRINT ;SETB TRANS INT FOR PLM51 STATUS CHECK
0059 209828   60  OVER: JB 98H,GCBACK ;IF RI NOT SET GCBACK
005C 0001     61  PUSH 01
005E A999     62  MOV R1,80H ;READ SBUF
0060 0298     63  CLN 098H ;CLEAR RI BIT
0062 0000     64  PUSH PSW ;PUSH REGISTERS USED BY PLM51
0064 00E0     65  PUSH ACC
0066 0000     66  PUSH 00H
0068 0200 F    67  CLN ESCSEQ ;CLR ESC SEQUENCE FLAG
006A 7400 F    68  MOV A,#SERIAL ;GET SERIAL FIFO RAM START LOCATION
006C 2500 F    69  ADD A,FIFC ;AND FIND HOW FAR INTO THE FIFO WE ARE
006E F800 70  MOV R0,A ;PUT IT INTO R0
006F E9       71  MOV R1,#00H
0070 02E7     72  CLN 0E7H ;CLR BIT 7 OF ACC
0072 F6       73  MOV 80H,A ;PUT DATA IN FIFO
0073 B41B02    74  CJNE A,#0H,OVER1 ;IF DATA IS NOT A ESC KEY THEN GO OVER
0075 0200 F    75  SETB ESCSEQ ;SET ESC SEQUENCE FLAG
0078 0500 F    76  OVER1: INC FIFC ;MOV FIFO TO NEXT LOCATION
007A 0200 F    77  SETB SERINT ;SET SERIAL INT BIT FOR PLM51 STATUS CHECK
007C 0000     78  POP 00H ;POP REGISTERS
007E 00E0     79  POP ACC
0080 0000     80  POP PSW
0082 0001     81  POP 01H
0084 32       82  GCBACK: RETI
0085 0000     83
0087 00E0     84  BLANK: PUSH PSW ;PUSH REG USED BY PLM51
0089 00E0     85  PUSH ACC
008B 00E0     86  PUSH DPL
008D 00E0     87  PUSH DPH
008F 0000     88  PUSH 00H
008F 850082 F  89  MOV DPL,LINE0+1 ;GET LINE0 INFO
0092 8500B3 F  90  MOV DPH,LINE0 ;AND PUT IT INTO DPH
0095 7850     91  MOV R0,#50H ;NUMBER OF CHARACTERS IN A LINE
0097 1420     92  NOTYET: MOV A,#20H ;ASCII SPACE CHARACTER
0099 F0       93  MOVX @DPTR,A ;MOV TO DISPLAY RAM
009A A3       94  INC DPTR ;INCH TO NEXT DISPLAY RAM LOCATION
009B 08FA     95  UJNZ R0,NOTYET ;IF ALL 50H LOCATIONS ARE NOT FILLED
009D 0000     96  ;GO DO MORE
009D 0000     97  POP 00H ;POP REGISTERS
009F 00E0     98  POP DPH
00A1 00E0     99  POP DPL
00A3 00E0    100  POP ACC
00A5 0000    101  POP PSW
00A7 22     102  RET
103 +1 SEJECT

```


MCS-51 MACRO ASSEMBLER

CRKASH

CRKASH

MCS-51 MACRO ASSEMBLER

```

LUC  UBJ      LINE  SOURCE
                                104
00A8 C0D0 ***** BLINK: PUSH PSW *****;PUSH REGISTERS USED BY PLM51
00AA C0E0 ***** PUSH ACC *****;PUSH REGISTERS USED BY PLM51
00AC C0B2 ***** PUSH LPL *****;PUSH REGISTERS USED BY PLM51
00AE C0B3 ***** PUSH UPH *****;PUSH REGISTERS USED BY PLM51
00B0 C0C0 ***** PUSH UOH *****;PUSH REGISTERS USED BY PLM51
00B2 8500B3 F 110 MOV UPH,PCIN1 ;GET CURRENT DISPLAY RAM LOCATION
00B5 8500B2 F 111 MOV UPL,PCINT+1 ;CALCULATED BY PLM51
00B8 43B31078 F 112 URL UPH,#10H ;SET BIT 15 FOR RAM ADDRESS DECODING
00BB 8B00 F 113 MOV R0,CURSEN ;GET CURSER COLUMN INFO TO TELL HOW
                                114
                                115 CONT1: MOV A,#20H ;ASCII SPACE CHARACTER
00BD 7420 81 116 MOVX @DPTN,A ;MOVE TO DISPLAY RAM
00BF F0 117 INC UPIR ;INCH TO NEXT DISPLAY RAM LOCATION
00C1 08 118 INC R0
00C2 8B50F8 119 CJNE R0,#50H,CONT1 ;IF NOT AT THE END OF THE LINE
                                120
                                121 POP UOH ;POP REGISTERS
00C7 00B3 122 POP UPH
00C9 00B2 123 POP UPL
00CB 00E0 124 POP ACC
00CD 00D0 125 POP PSW
00CF 22 126 RET
                                127
00D0 C0D0 128 FILL: PUSH PSW ;PUSH REGISTERS USED BY PLM51
00D2 C0E0 129 PUSH ACC
00D4 C0B2 130 PUSH UPL
00D6 C0B3 131 PUSH UPH
00D8 C0C0 132 PUSH UOH
00DA C3 133 CLR C
00DB 8500B3 F 134 MOV UPH,L ;GET BEGINNING OF LINE RAM LOCATION
00DE 8500B2 F 135 MOV UPL,L+1 ;CALCULATED BY PLM51
00E1 43B310 136 URL UPH,#10H ;SET BIT 15 FOR DISPLAY RAM ADDRESS DECODE
00E4 784F 137 MOV R0,#4FH ;SET UP COUNTER FOR 50H LOCATIONS
00E6 A3 138 INC UPIR ;GO PAST THE 0F1H
00E7 7420 139 CONT2: MOV A,#20H ;ASCII SPACE CHARACTER
00E9 F0 140 MOVX @DPTN,A ;MOVE TO DISPLAY RAM
00EA A3 141 INC UPIR ;INCH TO NEXT DISPLAY RAM LOCATION
00EB 0BF8 142 DJNZ R0,CUNT2 ;IF ALL 79 LOCATIONS HAVE NOT BEEN FILLED
                                143
                                144 POP UOH ;POP REGISTERS
00ED 00D0 145 POP UPH
00EF 00B3 146 POP UPL
00F1 00B2 147 POP ACC
00F3 00E0 148 POP PSW
00F5 00D0 149 RET
00F7 22 150
                                151
                                152 +1 SEJECT

```

MLS-51 MACRU ASSEMBLY

CXTASM

LUC	UPJ	LINE	SOURCE
1	1	1	1
2	2	2	2
3	3	3	3
4	4	4	4
5	5	5	5
6	6	6	6
7	7	7	7
8	8	8	8
9	9	9	9
10	10	10	10
11	11	11	11
12	12	12	12
13	13	13	13
14	14	14	14
15	15	15	15
16	16	16	16
17	17	17	17
18	18	18	18
19	19	19	19
20	20	20	20
21	21	21	21
22	22	22	22
23	23	23	23
24	24	24	24
25	25	25	25
26	26	26	26
27	27	27	27
28	28	28	28
29	29	29	29
30	30	30	30
31	31	31	31
32	32	32	32
33	33	33	33
34	34	34	34
35	35	35	35
36	36	36	36
37	37	37	37
38	38	38	38
39	39	39	39
40	40	40	40
41	41	41	41
42	42	42	42
43	43	43	43
44	44	44	44
45	45	45	45
46	46	46	46
47	47	47	47
48	48	48	48
49	49	49	49
50	50	50	50
51	51	51	51
52	52	52	52
53	53	53	53
54	54	54	54
55	55	55	55
56	56	56	56
57	57	57	57
58	58	58	58
59	59	59	59
60	60	60	60
61	61	61	61
62	62	62	62
63	63	63	63
64	64	64	64
65	65	65	65
66	66	66	66
67	67	67	67
68	68	68	68
69	69	69	69
70	70	70	70
71	71	71	71
72	72	72	72
73	73	73	73
74	74	74	74
75	75	75	75
76	76	76	76
77	77	77	77
78	78	78	78
79	79	79	79
80	80	80	80
81	81	81	81
82	82	82	82
83	83	83	83
84	84	84	84
85	85	85	85
86	86	86	86
87	87	87	87
88	88	88	88
89	89	89	89
90	90	90	90
91	91	91	91
92	92	92	92
93	93	93	93
94	94	94	94
95	95	95	95
96	96	96	96
97	97	97	97
98	98	98	98
99	99	99	99
100	100	100	100

Address	Hex	Assembly	Comment
154	154	LDI R0, 0	
155	155	LDI R1, 0	
156	156	LDI R2, 0	
157	157	LDI R3, 0	
158	158	LDI R4, 0	
159	159	LDI R5, 0	
160	160	LDI R6, 0	
161	161	LDI R7, 0	
162	162	LDI R8, 0	
163	163	LDI R9, 0	
164	164	LDI R10, 0	
165	165	LDI R11, 0	
166	166	LDI R12, 0	
167	167	LDI R13, 0	
168	168	LDI R14, 0	
169	169	LDI R15, 0	
170	170	LDI R16, 0	
171	171	LDI R17, 0	
172	172	LDI R18, 0	
173	173	LDI R19, 0	
174	174	LDI R20, 0	
175	175	LDI R21, 0	
176	176	LDI R22, 0	
177	177	LDI R23, 0	
178	178	LDI R24, 0	
179	179	LDI R25, 0	
180	180	LDI R26, 0	
181	181	LDI R27, 0	
182	182	LDI R28, 0	
183	183	LDI R29, 0	
184	184	LDI R30, 0	
185	185	LDI R31, 0	
186	186	LDI R32, 0	
187	187	LDI R33, 0	
188	188	LDI R34, 0	
189	189	LDI R35, 0	
190	190	LDI R36, 0	
191	191	LDI R37, 0	
192	192	LDI R38, 0	
193	193	LDI R39, 0	
194	194	LDI R40, 0	
195	195	LDI R41, 0	
196	196	LDI R42, 0	
197	197	LDI R43, 0	
198	198	LDI R44, 0	
199	199	LDI R45, 0	
200	200	LDI R46, 0	
201	201	LDI R47, 0	
202	202	LDI R48, 0	
203	203	LDI R49, 0	
204	204	LDI R50, 0	
205	205	LDI R51, 0	
206	206	LDI R52, 0	
207	207	LDI R53, 0	

PLS-51 MACRO ASSEMBLER

CMTASK

M2A1D

M2A1D MACRO ASSEMBLER

LUC	OBJ	LINE	SOURCE	SOURCE	LINE	LUC	OBJ
0130	A3	208	INC A	UPTR	10a	0a	1010
0131	E0	209	MOVX A, aDPTM		10b	1a	0010
0132	A3	210	INC A	UPTR	10c	0a	0010
0133	E0	211	MOVX A, aDPTM		10d	1a	0010
0134	A3	212	INC A	UPTR	10e	0a	0010
0135	E0	213	MOVX A, aDPTM		10f	1a	0010
0136	A3	214	INC A	UPTR	10g	0a	0010
0137	E0	215	MOVX A, aDPTM		10h	1a	0010
0138	A3	216	INC A	UPTR	10i	0a	0010
0139	E0	217	MOVX A, aDPTM		10j	1a	0010
013A	A3	218	INC A	UPTR	10k	0a	0010
013B	E0	219	MOVX A, aDPTM		10l	1a	0010
013C	A3	220	INC A	UPTR	10m	0a	0010
013D	E0	221	THIRTY: MOVX A, aDPTM		10n	1a	0010
013E	A3	222	INC A	UPTR	10o	0a	0010
013F	E0	223	MOVX A, aDPTM		10p	1a	0010
0140	A3	224	INC A	UPTR	10q	0a	0010
0141	E0	225	MOVX A, aDPTM		10r	1a	0010
0142	A3	226	INC A	UPTR	10s	0a	0010
0143	E0	227	MOVX A, aDPTM		10t	1a	0010
0144	A3	228	INC A	UPTR	10u	0a	0010
0145	E0	229	MOVX A, aDPTM		10v	1a	0010
0146	A3	230	INC A	UPTR	10w	0a	0010
0147	E0	231	MOVX A, aDPTM		10x	1a	0010
0148	A3	232	INC A	UPTR	10y	0a	0010
0149	E0	233	MOVX A, aDPTM		10z	1a	0010
014A	A3	234	INC A	UPTR	10A	0a	0010
014B	E0	235	MOVX A, aDPTM		10B	1a	0010
014C	A3	236	INC A	UPTR	10C	0a	0010
014D	E0	237	MOVX A, aDPTM		10D	1a	0010
014E	A3	238	INC A	UPTR	10E	0a	0010
014F	E0	239	MOVX A, aDPTM		10F	1a	0010
0150	A3	240	INC A	UPTR	10G	0a	0010
0151	E0	241	FORTY: MOVX A, aDPTM		10H	1a	0010
0152	A3	242	INC A	UPTR	10I	0a	0010
0153	E0	243	MOVX A, aDPTM		10J	1a	0010
0154	A3	244	INC A	UPTR	10K	0a	0010
0155	E0	245	MOVX A, aDPTM		10L	1a	0010
0156	A3	246	INC A	UPTR	10M	0a	0010
0157	E0	247	MOVX A, aDPTM		10N	1a	0010
0158	A3	248	INC A	UPTR	10O	0a	0010
0159	E0	249	MOVX A, aDPTM		10P	1a	0010
015A	A3	250	INC A	UPTR	10Q	0a	0010
015B	E0	251	MOVX A, aDPTM		10R	1a	0010
015C	A3	252	INC A	UPTR	10S	0a	0010
015D	E0	253	MOVX A, aDPTM		10T	1a	0010
015E	A3	254	INC A	UPTR	10U	0a	0010
015F	E0	255	MOVX A, aDPTM		10V	1a	0010
0160	A3	256	INC A	UPTR	10W	0a	0010
0161	E0	257	MOVX A, aDPTM		10X	1a	0010
0162	A3	258	INC A	UPTR	10Y	0a	0010
0163	E0	259	MOVX A, aDPTM		10Z	1a	0010
0164	A3	260	INC A	UPTR	10A	0a	0010
0165	E0	261	FIFTY: MOVX A, aDPTM		10B	1a	0010
0166	A3	262	INC A	UPTR	10C	0a	0010

LUC	OBJ	LINE	SOURCE	ADDRESS	VALUE	DISP	OBJ
0167	E0	263	MOVX A, @DPTM	000			2A 0110
0168	A3	264	INC A	001			00 0110
0169	E0	265	MOVX A, @DPTM	010			EA 0110
016A	A3	266	INC A	011			02 0110
016B	E0	267	MOVX A, @DPTM	012			EA 0110
016C	A3	268	INC A	013			02 0110
016D	E0	269	MOVX A, @DPTM	014			EA 0110
016E	A3	270	INC A	015			02 0110
016F	E0	271	MOVX A, @DPTM	016			EA 0110
0170	A3	272	INC A	017			02 0110
0171	E0	273	MOVX A, @DPTM	018			EA 0110
0172	A3	274	INC A	019			02 0110
0173	E0	275	MOVX A, @DPTM	020			EA 0110
0174	A3	276	INC A	021			02 0110
0175	E0	277	MOVX A, @DPTM	022			EA 0110
0176	A3	278	INC A	023			02 0110
0177	E0	279	MOVX A, @DPTM	024			EA 0110
0178	A3	280	INC A	025			02 0110
0179	E0	281	MOVX A, @DPTM	026			EA 0110
017A	A3	282	INC A	027			02 0110
017B	E0	283	MOVX A, @DPTM	028			EA 0110
017C	A3	284	INC A	029			02 0110
017D	E0	285	MOVX A, @DPTM	030			EA 0110
017E	A3	286	INC A	031			02 0110
017F	E0	287	MOVX A, @DPTM	032			EA 0110
0180	A3	288	INC A	033			02 0110
0181	E0	289	MOVX A, @DPTM	034			EA 0110
0182	A3	290	INC A	035			02 0110
0183	E0	291	MOVX A, @DPTM	036			EA 0110
0184	A3	292	INC A	037			02 0110
0185	E0	293	MOVX A, @DPTM	038			EA 0110
0186	A3	294	INC A	039			02 0110
0187	E0	295	MOVX A, @DPTM	040			EA 0110
0188	A3	296	INC A	041			02 0110
0189	E0	297	MOVX A, @DPTM	042			EA 0110
018A	A3	298	INC A	043			02 0110
018B	E0	299	MOVX A, @DPTM	044			EA 0110
018C	A3	300	INC A	045			02 0110
018D	E0	301	MOVX A, @DPTM	046			EA 0110
018E	A3	302	INC A	047			02 0110
018F	E0	303	MOVX A, @DPTM	048			EA 0110
0190	A3	304	INC A	049			02 0110
0191	E0	305	MOVX A, @DPTM	050			EA 0110
0192	A3	306	INC A	051			02 0110
0193	E0	307	MOVX A, @DPTM	052			EA 0110
0194	A3	308	INC A	053			02 0110
0195	E0	309	MOVX A, @DPTM	054			EA 0110
0196	A3	310	INC A	055			02 0110
0197	E0	311	MOVX A, @DPTM	056			EA 0110
0198	A3	312	INC A	057			02 0110
0199	E0	313	MOVX A, @DPTM	058			EA 0110
019A	A3	314	INC A	059			02 0110
019B	E0	315	MOVX A, @DPTM	060			EA 0110
019C	A3	316	INC A	061			02 0110
019D	E0	317	MOVX A, @DPTM	062			EA 0110

MCS-51 MACRO ASSEMBLER

CHTASM

ASBATHO

ASSEMBLER

SYMBOL TABLE LISTING

NAME	TYPE	VALUE	ATTRIBUTES
ACC. . . .	D ADDR	00E0H	A
BLANK. . . .	C ADDR	00E5H	A PUB
BLINE. . . .	C ADDR	00A8H	A PUB
BUFFER. . . .	C ADDR	003FH	A
CHECK. . . .	C ADDR	01A3H	A
CUN11. . . .	C ADDR	00B6H	A
CUN12. . . .	C ADDR	00E7H	A
COUNT. . . .	D ADDR	----	EXT
CURSER. . . .	D ADDR	----	EXT
ODONE. . . .	C ADDR	00F8H	A
DMA. . . .	C ADDR	00FAH	A
DMAONE. . . .	C ADDR	01B8H	A
DONE. . . .	C ADDR	01B4H	A
DPH. . . .	D ADDR	00E3H	A
DPL. . . .	D ADDR	00E2H	A
EIGHTY. . . .	C ADDR	01A1H	A
ESCSEW. . . .	FB ADDR	----	EXT
FIFU. . . .	FB ADDR	----	EXT
FIFTY. . . .	C ADDR	01E5H	A
FILL. . . .	C ADDR	00C0H	A PUB
FURTY. . . .	C ADDR	0151H	A
GUBACK. . . .	C ADDR	00E4H	A
L. . . .	D ADDR	----	EXT
LINEO. . . .	D ADDR	----	EXT
NUTYET. . . .	C ADDR	0097H	A
OVER. . . .	C ADDR	0059H	A
OVER1. . . .	C ADDR	0078H	A
POINT. . . .	D ADDR	----	EXT
PSW. . . .	D ADDR	00D0H	A
RASIEH. . . .	D ADDR	----	EXT
SBUF. . . .	D ADDR	0099H	A
SCAN. . . .	B ADDR	----	EXT
SERBUF. . . .	C ADDR	0052H	A
SERIAL. . . .	D ADDR	----	EXT
SERINT. . . .	B ADDR	----	EXT
SEVNTY. . . .	C ADDR	01E0H	A
SIXTY. . . .	C ADDR	0179H	A
TEN. . . .	C ADDR	0115H	A
THIRTY. . . .	C ADDR	0136H	A
THNINI. . . .	B ADDR	----	EXT
THENTY. . . .	C ADDR	0129H	A
VERI. . . .	C ADDR	0025H	A

REGISTER BANK(S) USED: 0

ASSEMBLY COMPLETE, NO ERRORS FOUND

MUS-51 MACRU ASSEMBLER

КЕУБД

ISIS-II MLS-51 PALRC ASSEMBLER v2.1

OBJECT MODULE PLACED IN :F1:KEYBCD.OPJ

ASSEMBLER INVOKED BY: ASF51 :F1:KEYBU.SHL

LUC	UBJ	LINE	SOURCE
1	1	1	1
2	2	2	2
3	3	3	3
4	4	4	4
5	5	5	5
6	6	6	6
7	7	7	7
8	8	8	8
9	9	9	9
10	10	10	10
11	11	11	11
12	12	12	12
13	13	13	13
14	14	14	14
15	15	15	15
16	16	16	16
17	17	17	17
18	18	18	18
19	19	19	19
20	20	20	20
21	21	21	21
22	22	22	22
23	23	23	23
24	24	24	24
25	25	25	25
26	26	26	26
27	27	27	27
28	28	28	28
29	29	29	29
30	30	30	30
31	31	31	31
32	32	32	32
33	33	33	33
34	34	34	34
35	35	35	35
36	36	36	36
37	37	37	37
38	38	38	38
39	39	39	39
40	40	40	40
41	41	41	41
42	42	42	42
43	43	43	43
44	44	44	44
45	45	45	45
46	46	46	46
47	47	47	47
48	48	48	48
49	49	49	49
50	50	50	50
51	51	51	51
52	52	52	52
53	53	53	53
54	54	54	54
55	55	55	55
56	56	56	56
57	57	57	57
58	58	58	58
59	59	59	59
60	60	60	60
61	61	61	61
62	62	62	62
63	63	63	63
64	64	64	64
65	65	65	65
66	66	66	66
67	67	67	67
68	68	68	68
69	69	69	69
70	70	70	70
71	71	71	71
72	72	72	72
73	73	73	73
74	74	74	74
75	75	75	75
76	76	76	76
77	77	77	77
78	78	78	78
79	79	79	79
80	80	80	80
81	81	81	81
82	82	82	82
83	83	83	83
84	84	84	84
85	85	85	85
86	86	86	86
87	87	87	87
88	88	88	88
89	89	89	89
90	90	90	90
91	91	91	91
92	92	92	92
93	93	93	93
94	94	94	94
95	95	95	95
96	96	96	96
97	97	97	97
98	98	98	98
99	99	99	99
100	100	100	100

```

1
2
3
4 ; *****
5 ; *****
6 ; ****
7 ; **** SOFTWARE FOR READING AN UNDECODED
8 ; **** KEYBOARD
9 ; ****
10 ; *****
11 ; *****
12 ;
13 ; GET KEYBOARD MAIN PORTIN
14 ; INITIALIZE LOOP COUNTER
15 ; INITIALIZE PLSM STATUS BITS
16 ; THIS CONTAINS THE SOFTWARE NEEDED TO SCAN AN UNDECODED KEYBOARD
17 ; THIS PROGRAM MUST BE LINKED TO THE MAIN PROGRAMS TO FUNCTION
18 ;
19 ;
20 ; MEMORY MAP FOR READING KEY BOARD (USING MOVCL)
21 ; IF SCAN HAS BEEN GO HIGHER THAN 0
22 ; LAST BIT IN THE SAME
23 ; OTHER CLR SAME BIT AND WRITE INFORMATION
24 ;
25 ; IF EQUAL JMP OVER TOP OF UP COUNTER
26 ; WHICH ZERO COUNTER
27 PUBLIC READEN
28 EXTRN DATA (LSTKEY)
29 EXTRN ITT (KEYO, SAME)
30 ; IF LOOP COUNTER NOT = SCANIN THERE 0
31 ; *****
32 ; IF YES SET RETO BIT
33 ; * "READEN" ROUTINE "
34 ; *
35 ; *****
36 +1 JEJECT

```

MCS-51 MACRO ASSEMBLER KEYBD

SYSTEM 83144308A 09JAN 12-03JN

LUC GRJ LINE SOURCE

1.5v 83144308A 09JAN 12-03JN 11-07c1
 1e0.00v31111 11 13JAN 13JUN 13c1e0
 JRD.08Y31111 1248A 116 13JAN 13JUN 13c1e0

```

3/
38 UNDECODED_KEYBOARD_SEGMENT CODE
39 KSEG UNDECODED_KEYBOARD
40
41
42
43 HEADER: PUSH PSW ;PUSH REG USED BY PLM51
44 PUSH ACC
45 ***** PUSH DPL *****
46 ***** PUSH DPH *****
47 PUSH 00H
48 ***** PUSH 00H *****
49 ***** PUSH 00H *****
50 ***** PUSH 00H *****
51 ***** MOV DPTR,#10FFH ***** INITIALIZE DPTR TO KEYBOARD
52 ***** ADDRESS *****
53 MOV R1,#00H ;CLR ZERO COUNTER
54 MOV R0,#LSIKEY ;GET KEYBOARD RAM POINTER
55 MOV R3,#08H ;INITIALIZE LOOP COUNTER
56 CLR KEY0 ;INITIALIZE PLM51 STATUS BITS
57 SETB SAME ;SAME
58 ***** MOV R2H,R02H ***** MOV LAST KEYBOARD SCAN TO 02H
59 CLM A
60 MOVC A,A+DPTH ;SCAN KEYBOARD
61 ***** CPL A ***** INVERT
62 JZ ZERO ;IF SCAN WAS ZERO GO INCREMENT ZERO COUNTER
63 CJNE A,R2H,NTSAME ;COMPARE WITH LAST SCAN IF NOT THE SAME
64 ;THEN CLR SAME BIT AND WRITE NEW INFORMATION
65 ;TO RAM
66 SJMP EQUAL ;IF EQUAL JMP OVER INCR OF ZERO COUNTER
67 ZERO: INC R1H ;INCR ZERO COUNTER
68 CJNE A,R2H,NTSAME ;SAME
69 EQUAL: INC R0 ;STEP TO NEXT SCAN RAM LOCATION
70 INC DPH ;NEXT KEYBOARD ADDRESS
71 DJNZ R3,MORE ;IF LOOP COUNTER NOT 0, SCAN AGAIN
72 CJNE R1,#08H,BACK ;CHECK TO SEE IF ALL 8 SCANS WHERE 0
73 SETB KEY0 ;IF YES SET KEY0 BIT
74 CLM SAME ;SAME
75 BACK: POP R3H
76 POP R2H ;POP REGISTERS
77 POP R1H
78 POP R0H
79 POP DPH
80 POP DPL
81 POP ACC
82 POP PSW
83 RET
84
85 ***** NTSAME: MOV R0,A ***** IF SCAN WAS NOT THE SAME THEN PUT NEW
86 ;SCAN INFO INTO RAM
87 CLM SAME ;CLR SAME BIT
88 SJMP EQUAL ;GO DO MORE
89
90
91 END

```


MCS-51 MACRO ASSEMBLER

KEYBD

304000 12-20M 12-20M

SYMBOL TABLE LISTING

12-20M 12-20M 12-20M
 12-20M 12-20M 12-20M
 12-20M 12-20M 12-20M

NAME	TYPE	VALUE	ATTRIBUTES
ACC.	D ADDR	00E0H	A
BACK.	C ADDR	003AH	R SEG=UNDECODED_KEYBOARD
DPH.	D ADDR	0083H	A
DPL.	D ADDR	0082H	A
EQUAL.	C ADDR	002EH	R SEG=UNDECODED_KEYBOARD
KEYU.	B ADDR	----	EXT
LSTKEY.	D ADDR	----	EXT
MORE.	C ADDR	0010H	R SEG=UNDECODED_KEYBOARD
NISAME.	C ADDR	0040H	R SEG=UNDECODED_KEYBOARD
PSW.	C ADDR	00D0H	A
READER.	C ADDR	0000H	R PUB. SEG=UNDECODED_KEYBOARD
SAME.	B ADDR	----	EXT
UNDECODED_KEYBOARD	C SEG	0050H	REL=LNII
ZERU.	C ADDR	0029H	R SEG=UNDECODED_KEYBOARD

REGISTER BANK(S) USED: 0

12-20M 12-20M 12-20M
 12-20M 12-20M 12-20M
 12-20M 12-20M 12-20M

ASSEMBLY COMPLETE, NO ERRORS FOUND

12-20M 12-20M 12-20M
 12-20M 12-20M 12-20M
 12-20M 12-20M 12-20M

MCS-51 MACRO ASSEMBLER DECODE

KEYBD 00000000 00000000 00000000 00000000

ISIS-11 MCS-51 MACRO ASSEMBLER V2.1
 OBJECT MODULE PLACED IN :F1:DECODE.OBJ
 ASSEMBLER INVOKED BY: ASP51 :F1:DECODE.SRC

SYMBOL TABLE LISTING
 =====

```

      1  LUC OBJ LINE SOURCE
      2
      3
      4 ;*****
      5 ;*****
      6 ;***
      7 ;***** SOFTWARE FOR DECODED KEYBOARD *****
      8 ;*****
      9 ;*****
     10 ;*****
     11 ;
     12 ;
     13 ;*****
     14 ;
     15 PUBLIC DETACH
     16 EXTRN DATA (LSIKEY)
     17 EXTRN BIT (ABDINT)
     18
     19
     20 ;
     21 ;*****
     22 ;*
     23 ;* "DECODE" INTERRUPT ROUTINE FOR DECODED KEYBOARDS *
     24 ;*
     25 ;*****
    26 +1 SEJECT
  
```

MCS-51 MACRO ASSEMBLER DECODE

MCS-51 MACRO ASSEMBLER DECODE

```

LCC  OBJ      LINE  SOURCE
-----
27
28 DECODED_KEYBOARD_SEGMENT CODE
29 MSG DECODED_KEYBOARD
30
31 DETACH: PUSH    PSH      ;PUSH REGISTERS
32           PUSH    CPL     ;USED BY PLM51
33           PUSH    LPH     ;
34           PUSH    ACC     ;
35           MOV     DPTR, #00FFH ;ADDRESS FOR KEYBOARD
36           CLR     A       ;
37           MOV     A, A+DPTR ;FETCH ASCII BYTE
38           MOV     LSKEY+1, A ;MOV TO MEMORY TO BE READ BY PLM51
39           SETB    ABINT    ;LET PLM51 KNOW THERE IS A BYTE
40           MOV     IP0, #0FFH ;SET COUNTER TO 0FFH SO INTERRUPT
41           MOV     TL0, #0FFH ;ON THE NEXT FALLING EDGE OF T0
42           POP     ACC
43           POP     DPH
44           POP     DPL
45           POP     PSH
46           RETI
47
48
49
50
51 END

```

SYMBOL TABLE LISTING

NAME	TYPE	VALUE	ATTRIBUTES
ACC.	D ADDR	00E0H	A
DECODED_KEYBOARD	C SEG	0020H	REL=LNII
DETACH	C ADDR	0000H	R PUB SEG=DECODED_KEYBOARD
DPH.	D ADDR	0083H	A
DPL.	D ADDR	0082H	A
KBDINT.	R ADDR	----	EXT
LSTKEY.	D ADDR	----	EXT
PSW.	D ADDR	00D0H	A
TH0.	D ADDR	008CH	A
TLO.	D ADDR	008AH	A

REGISTER BANK(S) USED: 0

ASSEMBLY COMPLETE, NO ERRORS FOUND


```

MCS-51 MACRO ASSEMBLER      DETACH      MCS-51 MACRO ASSEMBLER
                                nDAT30      12-50M

LUC OBJ      LINE      SOURCE

                                20      ;
                                21      ;
                                22
00B4          23      INPUT EQU TV
                                24
                                25
                                26      PUBLIC DETACH
                                27      EXTRN DATA (LSIKEY)
                                28      EXTRN BIT (RCVFLG, SYNC, BYFIN)
                                29      EXTRN BIT (KBDINI, ERKOK)
                                30
                                31 *****
                                32 *****
                                33 *****
                                34 *****
                                35 *****
                                36 *****
                                37 *****
                                38 *****
                                39 *****
                                40 *****
                                41 *****
                                42 *****
                                43 *****
                                44 *****
                                45 *****
                                46 *****
0000          47      START0      EQU      000H      ;LOW BYTE FOR 150 BAUD
00F4          48      START1      EQU      0F4H      ;HIGH BYTE FOR 150 BAUD
0000          49      MESSAGE0    EQU      000H      ;LOW BYTE FOR 150 BAUD
00E8          50      MESSAGE1    EQU      0E8H      ;HIGH BYTE FOR 150 BAUD
                                51 +1 $EJECT

```

MCS-51 MACRO ASSEMBLER DETACH

```

LUC OBJ          LINE    SOURCE
                    52
                    53
                    54 ;*****
                    55 ;*
                    56 ;*          "DETACH" INTERRUPT ROUTINE FOR DETACHABLE KEYBOARDS
                    57 ;*
                    58 ;*****
                    59
                    60
                    61
                    62 DETACHABLE_KEYBOARD SEGMENT CODE
                    63 NSEG DETACHABLE_KEYBOARD
                    64
-----
0000 C000          65 DETACH: PUSH    PSA                      ;PUSH REGISTERS USED BY PLM51
0002 C0C0          66          PUSH    ACC                      ;
0004 200013        F 67          JB      NCvFLG,VALID        ;IF RECEIVED FLAG SET GET NEXT BIT
0007 20B44A        68          JB      INPL1,NS1            ;IF T0 IS A 1 THEN NOT A START BIT
000A 0200          F 69          SETB   NCvFLG              ;IF T0 IS 0 THEN IT A START BIT
000C 758CF4        70          MOV     TPO,#SIANT1          ;SET TIMER TO INTERRUPT IN THE MIDDLE OF START BIT
000F 758A00        71          MOV     TLO,#SIANT0
0012 2589          72          MOV     A,TMUC
0014 22E2          73          CLK     UEZH                      ;SET TIMER COUNTER TO TIMER MODE
0016 F509          74          MOV     TMCD,A
0018 002B          75          SJMP    FINI                      ;GO BACK TO PROGRAM
                    76 ;
                    77 VALID:  JB      SYNC,NXTBIT1          ;CHECK IF VALID START BIT HAS BEEN SEEN
001D 20B434        F 78          JB      INPL1,NS1            ;IF NOT CHECK IF VALID START BIT
0020 0200          F 79          SETB   SYNC                ;IF YES SET SYNC
0022 750060        F 80          MOV     LSTKEY,#80H          ; INIT LSTKEY
0025 758CE8        81          MOV     TH0,#MESSAGE1          ;LOW NEXT BITTING CODE OF 10
0028 758A00        82          MOV     TLO,#MESSAGE0          ;SET TIMER FOR 1 BIT TIME
002B 0018          83          SJMP    FINI                      ;AND GO BACK TO MAIN PROGRAM
002C 0000          84 ;
002D 758CE8        85          NXTBIT: MOV    TH0,#MESSAGE1          ;
0030 758A00        86          MOV     TLO,#MESSAGE0          ;
0033 200014        F 87          JB      BYFIN,STOP          ;SET TIMER FOR 1 BIT TIME
0036 2500          F 88          MOV     A,LSTKEY          ;CHECK TO SEE IF ALL 8 BITS HAVE BEEN RECEIVED
0038 A204          89          MOV     C,INPL1          ;GET WORKING REGISTER
003A 13           90          KRC      A                      ;GET NEXT BIT FROM T1
003B F500          F 91          MOV     LSTKEY,A
003D 0006          92          JNC     FINI1
003F 0200          F 93          SETB   BYFIN
0041 22E7          94          CLK     UEZH
0043 F500          F 95          MOV     LSTKEY+1,A
0045 00E0          96          FINI1: POP     ACC
0047 0000          97          POP     PSA
0049 32           98          RETI

```

MCS-51 MACRO ASSEMBLER		DETACH	
0000	0000	29	WELL
0001	0000	31	WELL
LUC	0030	LINE	SOURCE
0002	0000	32	WELL
0003	0000	33	WELL
0004	300405	100	STOP:
0005	0200	F 101	SETB
0006	020000	F 102	JMP
0007	02	103	;
0008	0200	F 104	ERR:
0009	0200	F 105	RSI:
0010	0200	F 106	CLR
0011	0200	F 107	CLR
0012	0200	F 108	MOV
0013	0200	F 109	SETB
0014	0200	F 110	MOV
0015	758CFF	111	MOV
0016	758AFF	112	MOV
0017	800000	113	SJMP
0018	0500	114	SETB
0019	000000	115	;
0020	000000	116	AVG:
0021	000000	117	;
0022	0000	118	END
0023	0000	119	WELL
0024	0000	120	WELL
0025	0000	121	WELL
0026	0000	122	WELL
0027	0000	123	WELL
0028	0000	124	WELL
0029	0000	125	WELL
0030	0000	126	WELL
0031	0000	127	WELL
0032	0000	128	WELL
0033	0000	129	WELL
0034	0000	130	WELL
0035	0000	131	WELL
0036	0000	132	WELL
0037	0000	133	WELL
0038	0000	134	WELL
0039	0000	135	WELL
0040	0000	136	WELL
0041	0000	137	WELL
0042	0000	138	WELL
0043	0000	139	WELL
0044	0000	140	WELL
0045	0000	141	WELL
0046	0000	142	WELL
0047	0000	143	WELL
0048	0000	144	WELL
0049	0000	145	WELL
0050	0000	146	WELL
0051	0000	147	WELL
0052	0000	148	WELL
0053	0000	149	WELL
0054	0000	150	WELL
0055	0000	151	WELL
0056	0000	152	WELL
0057	0000	153	WELL
0058	0000	154	WELL
0059	0000	155	WELL
0060	0000	156	WELL
0061	0000	157	WELL
0062	0000	158	WELL
0063	0000	159	WELL
0064	0000	160	WELL
0065	0000	161	WELL
0066	0000	162	WELL
0067	0000	163	WELL
0068	0000	164	WELL
0069	0000	165	WELL
0070	0000	166	WELL
0071	0000	167	WELL
0072	0000	168	WELL
0073	0000	169	WELL
0074	0000	170	WELL
0075	0000	171	WELL
0076	0000	172	WELL
0077	0000	173	WELL
0078	0000	174	WELL
0079	0000	175	WELL
0080	0000	176	WELL
0081	0000	177	WELL
0082	0000	178	WELL
0083	0000	179	WELL
0084	0000	180	WELL
0085	0000	181	WELL
0086	0000	182	WELL
0087	0000	183	WELL
0088	0000	184	WELL
0089	0000	185	WELL
0090	0000	186	WELL
0091	0000	187	WELL
0092	0000	188	WELL
0093	0000	189	WELL
0094	0000	190	WELL
0095	0000	191	WELL
0096	0000	192	WELL
0097	0000	193	WELL
0098	0000	194	WELL
0099	0000	195	WELL
0100	0000	196	WELL
0101	0000	197	WELL
0102	0000	198	WELL
0103	0000	199	WELL
0104	0000	200	WELL
0105	0000	201	WELL
0106	0000	202	WELL
0107	0000	203	WELL
0108	0000	204	WELL
0109	0000	205	WELL
0110	0000	206	WELL
0111	0000	207	WELL
0112	0000	208	WELL
0113	0000	209	WELL
0114	0000	210	WELL
0115	0000	211	WELL
0116	0000	212	WELL
0117	0000	213	WELL
0118	0000	214	WELL
0119	0000	215	WELL
0120	0000	216	WELL
0121	0000	217	WELL
0122	0000	218	WELL
0123	0000	219	WELL
0124	0000	220	WELL
0125	0000	221	WELL
0126	0000	222	WELL
0127	0000	223	WELL
0128	0000	224	WELL
0129	0000	225	WELL
0130	0000	226	WELL
0131	0000	227	WELL
0132	0000	228	WELL
0133	0000	229	WELL
0134	0000	230	WELL
0135	0000	231	WELL
0136	0000	232	WELL
0137	0000	233	WELL
0138	0000	234	WELL
0139	0000	235	WELL
0140	0000	236	WELL
0141	0000	237	WELL
0142	0000	238	WELL
0143	0000	239	WELL
0144	0000	240	WELL
0145	0000	241	WELL
0146	0000	242	WELL
0147	0000	243	WELL
0148	0000	244	WELL
0149	0000	245	WELL
0150	0000	246	WELL
0151	0000	247	WELL
0152	0000	248	WELL
0153	0000	249	WELL
0154	0000	250	WELL
0155	0000	251	WELL
0156	0000	252	WELL
0157	0000	253	WELL
0158	0000	254	WELL
0159	0000	255	WELL
0160	0000	256	WELL
0161	0000	257	WELL
0162	0000	258	WELL
0163	0000	259	WELL
0164	0000	260	WELL
0165	0000	261	WELL
0166	0000	262	WELL
0167	0000	263	WELL
0168	0000	264	WELL
0169	0000	265	WELL
0170	0000	266	WELL
0171	0000	267	WELL
0172	0000	268	WELL
0173	0000	269	WELL
0174	0000	270	WELL
0175	0000	271	WELL
0176	0000	272	WELL
0177	0000	273	WELL
0178	0000	274	WELL
0179	0000	275	WELL
0180	0000	276	WELL
0181	0000	277	WELL
0182	0000	278	WELL
0183	0000	279	WELL
0184	0000	280	WELL
0185	0000	281	WELL
0186	0000	282	WELL
0187	0000	283	WELL
0188	0000	284	WELL
0189	0000	285	WELL
0190	0000	286	WELL
0191	0000	287	WELL
0192	0000	288	WELL
0193	0000	289	WELL
0194	0000	290	WELL
0195	0000	291	WELL
0196	0000	292	WELL
0197	0000	293	WELL
0198	0000	294	WELL
0199	0000	295	WELL
0200	0000	296	WELL
0201	0000	297	WELL
0202	0000	298	WELL
0203	0000	299	WELL
0204	0000	300	WELL
0205	0000	301	WELL
0206	0000	302	WELL
0207	0000	303	WELL
0208	0000	304	WELL
0209	0000	305	WELL
0210	0000	306	WELL
0211	0000	307	WELL
0212	0000	308	WELL
0213	0000	309	WELL
0214	0000	310	WELL
0215	0000	311	WELL
0216	0000	312	WELL
0217	0000	313	WELL
0218	0000	314	WELL
0219	0000	315	WELL
0220	0000	316	WELL
0221	0000	317	WELL
0222	0000	318	WELL
0223	0000	319	WELL
0224	0000	320	WELL
0225	0000	321	WELL
0226	0000	322	WELL
0227	0000	323	WELL
0228	0000	324	WELL
0229	0000	325	WELL
0230	0000	326	WELL
0231	0000	327	WELL
0232	0000	328	WELL
0233	0000	329	WELL
0234	0000	330	WELL
0235	0000	331	WELL
0236	0000	332	WELL
0237	0000	333	WELL
0238	0000	334	WELL
0239	0000	335	WELL
0240	0000	336	WELL
0241	0000	337	WELL
0242	0000	338	WELL
0243	0000	339	WELL
0244	0000	340	WELL
0245	0000	341	WELL
0246	0000	342	WELL
0247	0000	343	WELL
0248	0000	344	WELL
0249	0000	345	WELL
0250	0000	346	WELL
0251	0000	347	WELL
0252	0000	348	WELL
0253	0000	349	WELL
0254	0000	350	WELL
0255	0000	351	WELL
0256	0000	352	WELL
0257	0000	353	WELL
0258	0000	354	WELL
0259	0000	355	WELL
0260	0000	356	WELL
0261	0000	357	WELL
0262	0000	358	WELL
0263	0000	359	WELL
0264	0000	360	WELL
0265	0000	361	WELL
0266	0000	362	WELL
0267	0000	363	WELL
0268	0000	364	WELL
0269	0000	365	WELL
0270	0000	366	WELL
0271	0000	367	WELL
0272	0000	368	WELL
0273	0000	369	WELL
0274	0000	370	WELL
0275	0000	371	WELL
0276	0000	372	WELL
0277	0000	373	WELL

SYMBOL TABLE LISTING

NAME	TYPE	VALUE	ATTRIBUTES
ACC	C ADDR	00E0H	A
BYFIA	B ADDR	----	EXT
DETACH	C ADDR	0000H	N PUB
DETACHABLE_KEYBOARD	C SEG	0068H	SEG=DETACHABLE_KEYBOARD
ENR	C ADDR	0052H	N
ENRUR	B ADDR	----	EXT
FINI	C ADDR	0045H	N
INPUT	B ADDR	00B0H.4	A
KBDINT	B ADDR	----	EXT
LSTKEY	B ADDR	----	EXT
MESSAGE0	NLMB	0000H	A
MESSAGE1	NLMB	00E8H	A
NXTBIT	C ADDR	0020H	N
PSW	B ADDR	0000H	A
RCVFLG	B ADDR	----	EXT
RST	C ADDR	0054H	R
STAKTU	NLMB	0000H	A
STAKTI	NLMB	00F4H	A
STOP	C ADDR	004AH	R
SYNC	B ADDR	----	EXT
TU	B ADDR	00B0H.4	A
TH0	B ADDR	008CH	A
TLO	B ADDR	008AH	A
TMOU	B ADDR	0089H	A
VALID	C ADDR	001AH	R

REGISTER BANK(S) USED: 0

ASSEMBLY COMPLETE, NO ERRORS FOUND

1. John Minsky and George Alexy, CRT Terminal Design Using The Intel 8255, Intel Application Note AP-62, Nov., 1979.

2. John Minsky, A Low Cost CRT Terminal Using The Intel 8255, Intel Application Note AP-62, Nov., 1979.

When the address and RD/WR information to the 82786, which runs a cycle and deposits the 16 bit result into the latching transceivers at the end of the read cycle, as indicated by SEN. This event clears the BUSY flip flop in the EPLD. When the BUSY signal goes inactive, the 8051 reads the low byte from the latching transceiver at address C000H and the high byte from address C001H.

For write cycles, the 8051 writes the low byte of the word into the latch at address C000H and the high byte into address C001H. Next the upper address bits are set with PORT1 and a dummy write cycle is performed in graphics memory at the desired address (A000H-BFFFH). Like in the read example, the 82786 runs a memory cycle at this point, enabling the outputs of the latching transceivers at the proper time in the write cycle as indicated by SEN going active.

Accessing the registers inside the 82786 is done in exactly the same fashion, except that the 82786 is addressed in locations 8000H through 807EH. This causes the EPLD to drive the M/IO pin low during these cycles.

DESIGN NOTES

74F243's are used for the latching transceivers in this design. Although 74HCT04's could be used in a reduced pin count design, the 74F243's are preferred for the 82786 and 8051. The 74F243's are used for the 82786 and 8051. All addresses generated by the 8051 must be even byte addresses; the only byte operations allowed are the reads and writes to the latching transceivers. The design shown here incorporates hardware workarounds for the earlier "C-step" 82786; the current "D-step" part will work in the design as well. Additional information regarding the 82786 can be found in the "82786 Graphics Coprocessor User's Manual," Intel publication number 211933.

Interfacing the 82786 to the 8051 presents some interesting challenges. It can be accomplished with a little additional logic and software. Since the 82786 looks like a DRAM controller to the host CPU, wait states are often required when accessing the coprocessor. Since wait states are not supported by the 8051, latching transceivers and dummy read and write cycles are used to communicate with the 82786. Byte swapping is also required in the external logic to allow the 8 bit 8051 to read and write the 16 bit graphics memory supported by the 82786. This byte swapping is accomplished with the latching transceivers as well. All of the control logic is implemented in an Intel 80C00 EPLD, allowing the entire interface to fit into three 24 pin DIPs.

December 1987

HARDWARE

Figure 1 shows the interface between the 8051 and the 82786. Figure 2 shows a typical 8051 CPU design needed to complete the circuit. In this design the 82786 is mapped into an 8K byte window in 8051 data memory space. The upper address bits are used as a "page select", and are provided by I/O pins on the 8051. The 80C00 EPLD contains the control logic for the transceiver and address decoding for the 82786. An equivalent circuit for the EPLD is shown in Figure 3. The "LATCH" pin is shown in Figure 4. The 82786 data memory is mapped into 8K byte windows in 8051 data memory. The 8051 sets the upper address bits (PORT 1.0-1.3) and then performs a dummy read operation to the desired location in graphics memory (A000H-BFFFH). The dummy read cycle provides a third block of

OPERATION

Operation of the interface is as follows. For reading the graphics memory, the 8051 sets the upper address bits (PORT 1.0-1.3) and then performs a dummy read operation to the desired location in graphics memory (A000H-BFFFH). The dummy read cycle provides a third block of

RICK SCHUE
REGIONAL APPLICATIONS SPECIALIST
INDIANAPOLIS, INDIANA

Interfacing the 82786 to the 8051 presents some interesting challenges, but can be accomplished with a little additional logic and software. Since the 82786 looks like a DRAM controller to the host CPU, wait states are often required when accessing the coprocessor. Since wait states are not supported by the 8051, latching transceivers and dummy read and write cycles are used to communicate with the 82786. Byte swapping is also required in the external logic to allow the 8 bit 8051 to read and write the 16 bit graphics memory supported by the 82786. This byte swapping is accomplished with the latching transceivers as well. All of the control logic is implemented in an Intel 5C060 EPLD, allowing the entire interface to fit into three 24 pin DIPs.

HARDWARE

Figure 1 shows the interface between the 8051 bus and the 82786. Figure 2 shows a typical 8051 CPU design needed to complete the circuit. In this design the 82786 is mapped into an 8K byte window in 8051 data memory space. The upper address bits are used as a "page select" and are provided by I/O pins on the 8051. The 5C060 EPLD contains the control logic for the transceivers and address decoding for the 82786. An equivalent circuit for the EPLD is shown in Figure 3; the ".ADF" file is shown in Figure 4. The 82786 data memory is mapped into one 8K block (A000H-BFFE H), the 82786 registers are mapped into another (8000H-807EH), and the transceivers are mapped into a third block of memory (C000H-C001H).

OPERATION

Operation of the interface is as follows. For reading the graphics memory, the 8051 sets the upper address bits (PORT 1.0-1.3) and then performs a dummy read operation to the desired location in graphics memory (A000H thru BFFE H). The dummy read cycle pro-

vides the address and RD/WR information to the 82786, which runs a cycle and deposits the 16 bit result into the latching transceivers at the end of the read cycle, as indicated by SEN. This event clears the BUSY flip flop in the EPLD. When the BUSY signal goes inactive, the 8051 reads the low byte from the latching transceiver at address C000H and the high byte free address C001H.

For write cycles, the 8051 writes the low byte of the word into the latch at address C000H and the high byte into address C001H. Next the upper address bits are set with PORT1 and a dummy write cycle is performed in graphics memory at the desired address (A000H-BFFE H). Like in the read example, the 82786 runs a memory cycle at this point, enabling the outputs of the latching transceivers at the proper time in the write cycle, as indicated by SEN going active.

Accessing the registers inside the 82786 is done in exactly the same fashion, except that the 82786 is addressed in locations 8000H through 807EH. This causes the EPLD to drive the M/I/O pin low during these cycles.

DESIGN NOTES

74F543's are used for the latching transceivers in this design, although 74HCT646's could be used to reduce the total power consumption. Some changes to the EPLD would be required in this case. The interface assumes that all memory accesses to the 82786 are word references; accordingly, BHE is grounded at the 82786. All addresses generated by the 8051 must be even byte addresses, the only byte operations allowed are the reads and writes to the latching transceivers. The design shown here incorporates hardware work-arounds for the earlier "C-step" 82786; the current "D-step" part will work in the design as well. Additional information regarding the 82786 can be found in the "82786 Graphics Coprocessor User's Manual", Intel publication number 231933.

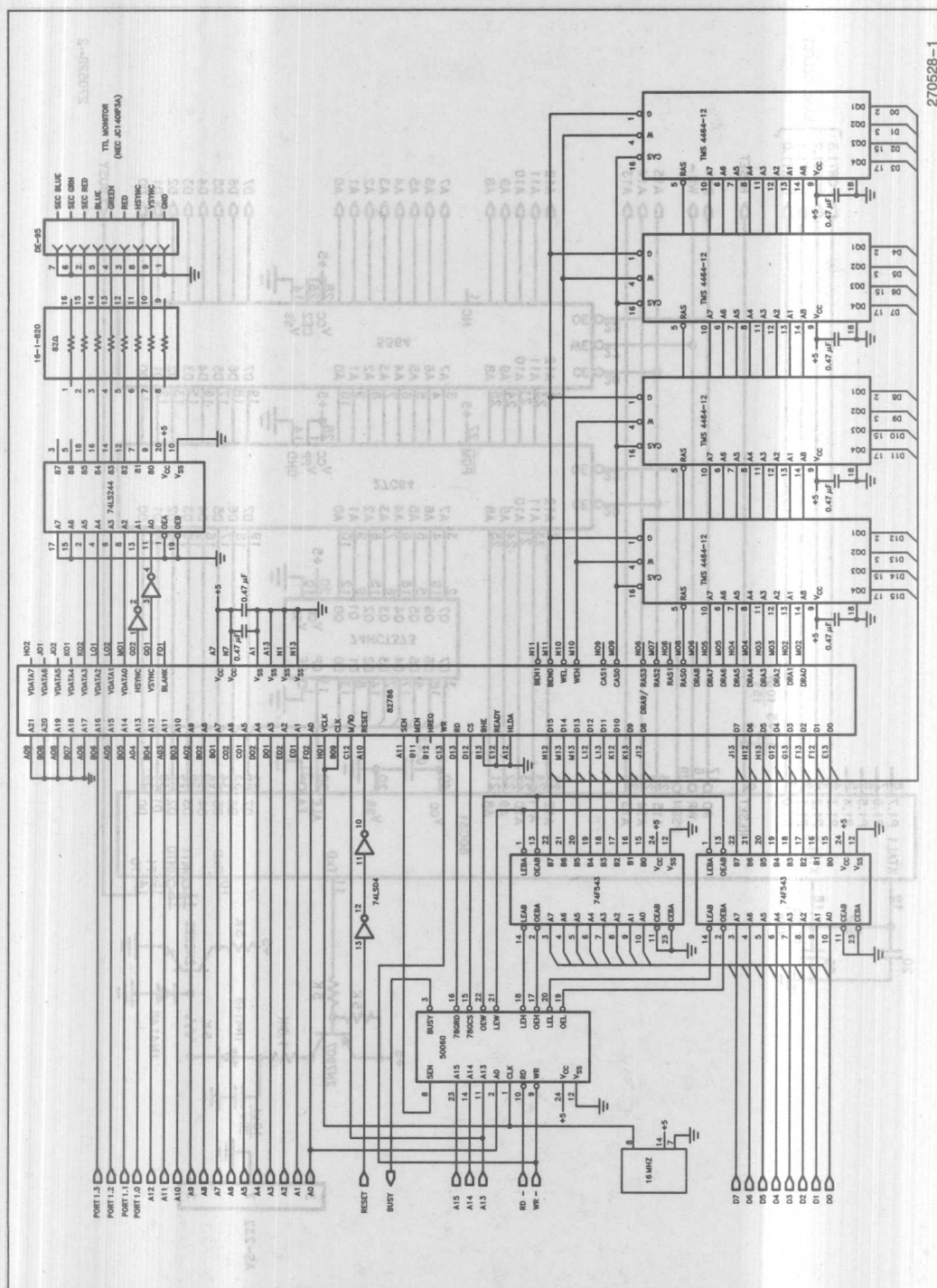


Figure 1
2-155

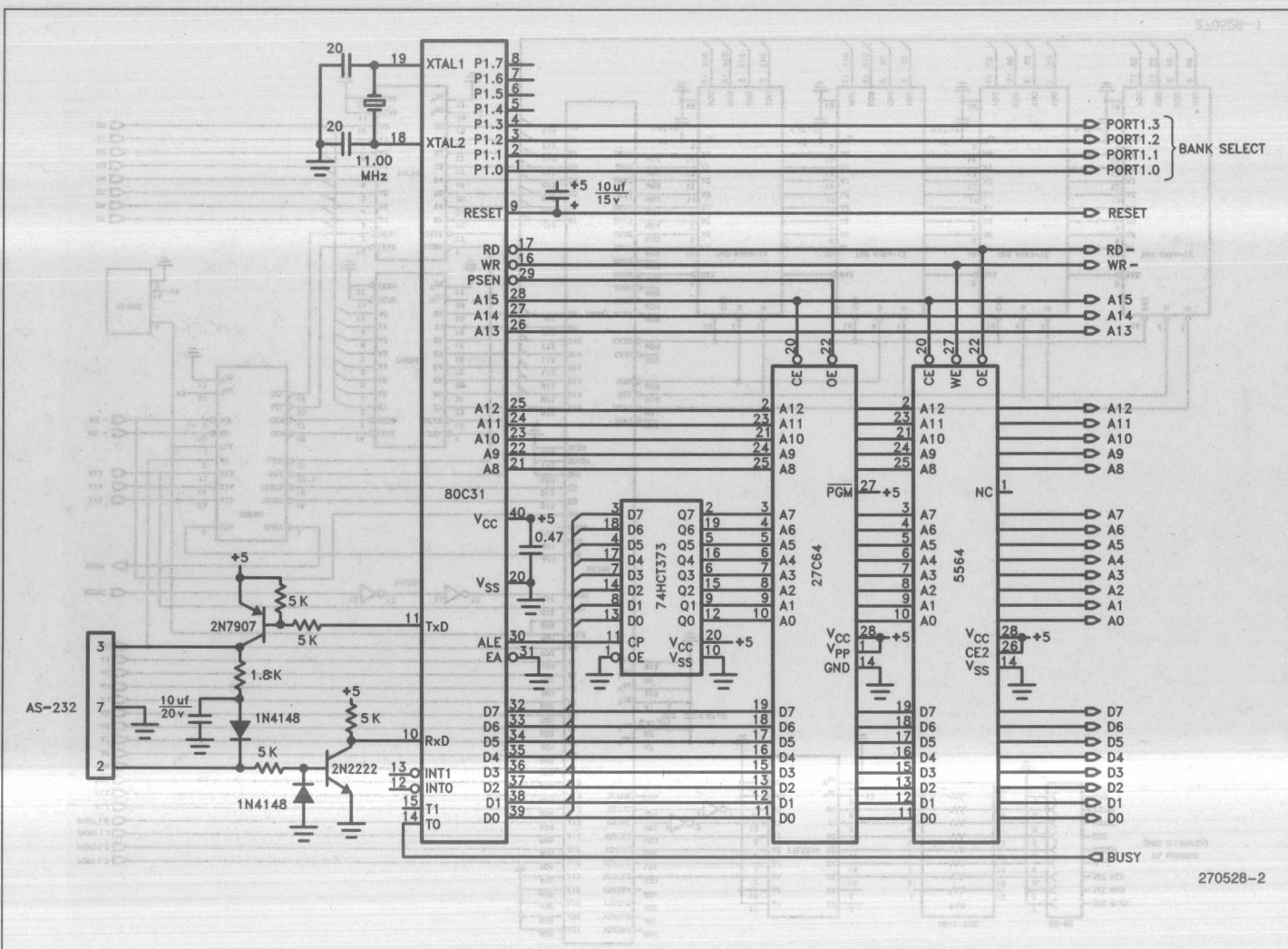


Figure 2
2-156

270528-2

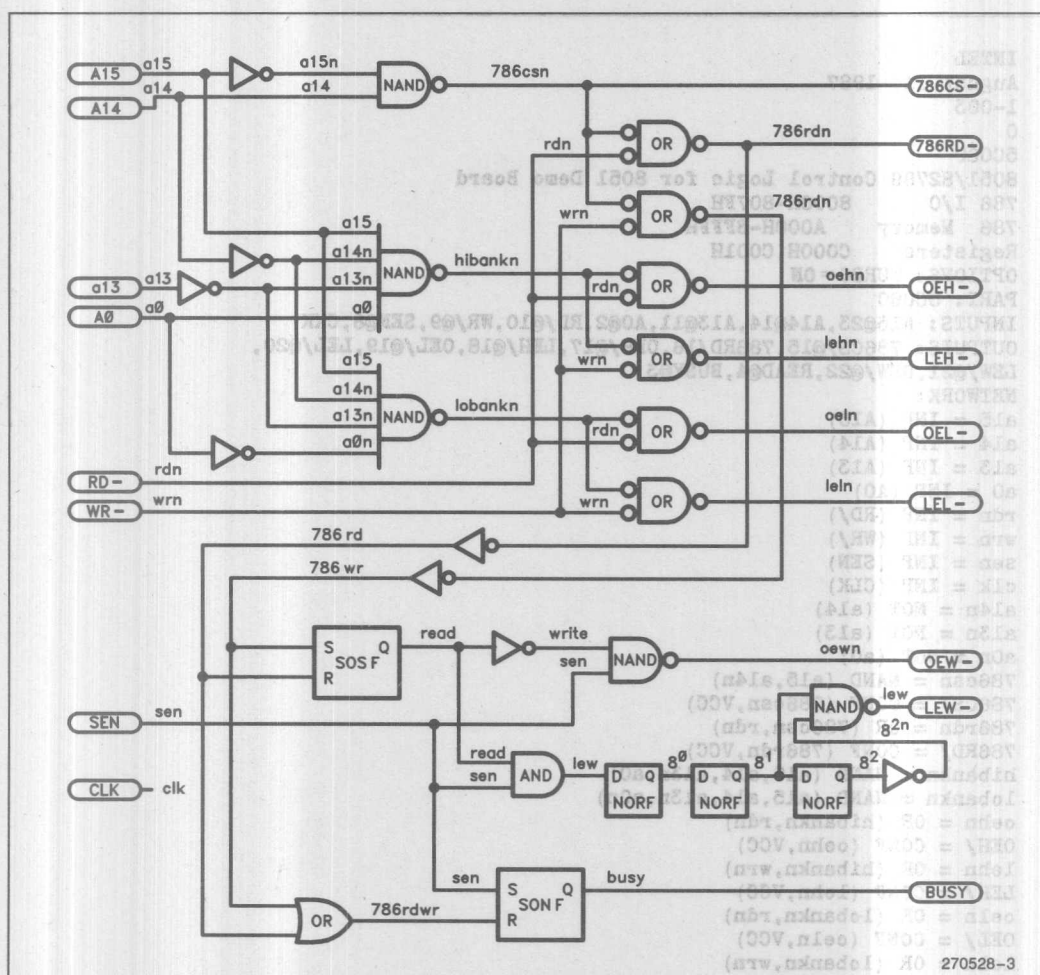


Figure 3. 8051/82786 Control EPLD (5C060-55) Equivalent Circuit

INTEL
August 11, 1987
1-003
0

5C060

8051/82786 Control Logic for 8051 Demo Board

786 I/O 8000H-807FH

786 Memory A000H-BFFFH

Registers C000H,C001H

OPTIONS: TURBO=ON

PART: 5C060

INPUTS: A15@23,A14@14,A13@11,A0@2,RD/@10,WR/@9,SEN@3,CLK

OUTPUTS: 786CS/@15,786RD/16,OEH/@17,LEH/@18,OEL/@19,LEL/@20,

LEW/@21,OEW/@22,READ@4,BUSY@3

NETWORK:

a15 = INP (A15)

a14 = INP (A14)

a13 = INP (A13)

a0 = INP (A0)

rdn = INP (RD/)

wrn = INP (WR/)

sen = INP (SEN)

clk = INP (CLK)

a14n = NOT (a14)

a13n = NOT (a13)

a0n = NOT (a0)

786csn = NAND (a15,a14n)

786CS/ = CONF (786csn,VCC)

786rdn = OR (786csn,rdn)

786RD/ = CONF (786rdn,VCC)

hibankn = NAND (a15,a14,a13n,a0)

lobankn = NAND (a15,a14,a13n,a0n)

oehn = OR (hibankn,rdn)

OEH/ = CONF (oehn,VCC)

lehn = OR (hibankn,wrn)

LEH/ = CONF (lehn,VCC)

oeln = OR (lobankn,rdn)

OEL/ = CONF (oeln,VCC)

leln = OR (lobankn,wrn)

LEL/ = CONF (leln,VCC)

oewn = NAND (sen,write)

OEW/ = CONF (oewn,VCC)

lew = AND (sen,read)

q0 = NORF (lew,clk,GND,GND)

q1 = NORF (q0,clk,GND,GND)

q2 = NORF (q1,clk,GND,GND)

q2n = NOT (q2)

lewn = NAND (q1,q2n)

LEW/ = CONF (lewn,VCC)

786wr = NOR (786csn,wrn)

786rd = NOT (786rdn)

READ,read = SOSF (786rd,clk,786wr,GND,GND,VCC)

write = NOT (read)

786rdwr = OR (786rd,786wr)

BUSY = SONF (786rdwr,clk,sen,GND,GND,VCC)

END\$

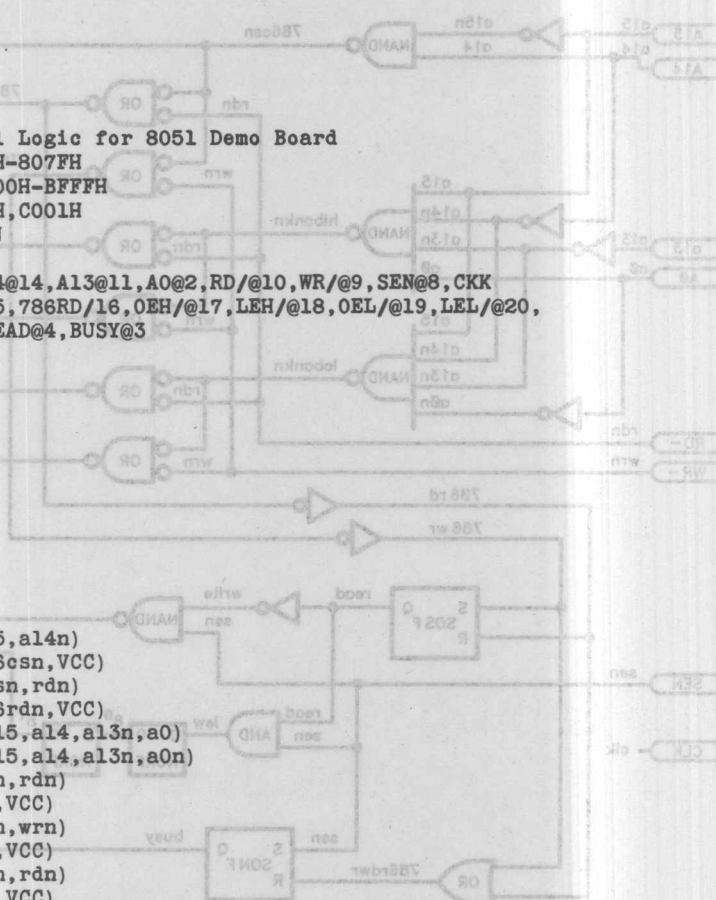


Figure 4.

Undefined results may occur if the software attempts to read address 8000H or 8001H, or write to address 8002H or 8003H.

TIMING REQUIREMENTS

The timing requirements of the Densitron LCD are a little slow for a full speed 8003H. The critical timing parameters are the enable pulse width (PW), E, of 450 ns and the data delay time during read cycles (tDDR) of 330 ns. The 8003H is available at clock speeds up to 16 MHz, but at this speed these parameters are violated. Since the 8003H lacks a READY pin, the only way to satisfy the LCD timing requirements is to slow the clock down to 10 MHz or lower. A convenient crystal frequency is 3.125 MHz, since it allows standard baud rates to be generated with the internal timer.

SOFTWARE

The code consists of a main module and a set of utility procedures that talk directly to the LCD. This way the application code does not have to be concerned with where the LCD is mapped, or the exact pin patterns needed to control it. The mainline consists of a call to initialize the LCD, and then it writes a message to the screen, waits, and then erases it. It repeats this indefinitely. The utility procedures include functions to initialize the LCD, send data and address to the LCD, home the cursor, set the display, set the cursor to a given row and column, turn the cursor on and off, and print a string of characters to the display. Not all the functions are used in the software example.

REFERENCES

- DENSITRON Catalog LCDMD-C
- INTEL P1/M-81 User's Guide, 121966
- INTEL Embedded Controller Handbook, 210918

RICK SCHUE
REGIONAL APPLICATIONS SPECIALIST
INDIANAPOLIS, INDIANA

INTRODUCTION

This application note details the interface between an 8003H and a Densitron two row by 24 character LM3232C2BW display. This combination provides a very flexible display format (2x24) and a cost effective, low power consumption microcontroller suitable for many industrial control and monitoring functions.

Although this application brief concentrates on the 8003H, the same software and hardware techniques are equally valid on other members of the 8031 family, including the 8031, 8032, and the 8044.

December 1987

HARDWARE DESIGN

The LCD is mapped into external data memory, and looks to the 8003H just like ordinary RAM. The register select (RS) and read/write (R/W) pins are connected to the low order address lines A0 and A1. Connecting the R/W pin to an address line is a little unconventional, but since the R/W line has the same set-up time requirements as the RS line, treating the R/W pin as an address keeps this pin from causing any timing problems.

The enable (E) pin of the LCD is used to select the device, and is connected to the 8003H's RD pin. The RD pin is active low, and the LCD's enable pin is active high. If this seems a little odd, it's because the LCD's enable pin is connected to a more elaborate address decoder.

With the address encoding shown in the example, the LCD is mapped as follows:

Address	Function	Read/Write?
8000H	Write Command to LCD	Write Only
8001H	Write Data to LCD	Write Only
8002H	Read Status from LCD	Read Only
8003H	Read Data from LCD	Read Only
8004H	to Access	
FFFFH		

INTRODUCTION

This application note details the interface between an 80C31 and a Densitron two row by 24 character LM23A2C24CBW display. This combination provides a very flexible display foot format (2x24) and a cost effective, low power consumption microcontroller suitable for many industrial control and monitoring functions.

Although this applications brief concentrates on the 80C31, the same software and hardware techniques are equally valid on other members of the 8051 family, including the 8031, 8751, and the 8044.

HARDWARE DESIGN

The LCD is mapped into external data memory, and looks to the 80C31 just like ordinary RAM. The register select (RS) and the read/write (R/W) pins are connected to the low order address lines A0 and A1. Connecting the R/W pin to an address line is a little unorthodox, but since the R/W line has the same set-up time requirements as the RS line, treating the R/W pin as an address kept this pin from causing any timing problems.

The enable (E) pin of the LCD is used to select the device, and is driven by the logical OR of the 80C31's RD and WR signals AND'ed with the MSB of the address bus. This maps the LCD into the upper half of the 64 KB external data space. If this seems a little wasteful, feel free to use a more elaborate address decoding scheme.

With the address decoding shown in the example, the LCD is mapped as follows:

Address	Function	Read/Write?
8000H	Write Command to LCD	Write Only
8001H	Write Data to LCD	Write Only
8002H	Read Status from LCD	Read Only
8003H	Read Data from LCD	Read Only
8004H to FFFFH	No Access	

Undefined results may occur if the software attempts to read address 8000H or 8001H, or write to address 8002H or 8003H.

TIMING REQUIREMENTS

The timing requirements of the Densitron LCD are a little slow for a full speed 80C31. The critical timing parameters are the enable pulse width (PW E) of 450 ns, and the data delay time during read cycles (tDDR) of 320 ns. The 80C31 is available at clock speeds up to 16 MHz, but at this speed these parameters are violated. Since the 80C31 lacks a READY pin, the only way to satisfy the LCD timing requirements is to slow the clock down to 10 MHz or lower. A convenient crystal frequency is 7.3728 MHz since it allows all standard baud rates to be generated with the internal timers.

SOFTWARE

The code consists of a main module and a set of utility procedures that talk directly to the LCD. This way the application code does not have to be concerned with where the LCD is mapped, or the exact bit patterns needed to control it. The mainline consists of a call to initialize the LCD, and then it writes a message to the screen, waits, and then erases it. It repeats this indefinitely.

The utility procedures include functions to initialize the display, send data and ads to the LCD, home the cursor, clear the display, set the cursor to a given row and column, turn the cursor on and off, and print a string of characters to the display. Not all the functions are used in the software example.

REFERENCES

INTEL Embedded Controller Handbook, 210918

INTEL PL/M-51 User's Guide, 121966

DENSITRON Catalog LCDMD-C


```

Main_module: DO;

Delay: PROCEDURE (count) EXTERNAL;
  DECLARE    count          WORD;
END Delay;

Initialize_LCD: PROCEDURE EXTERNAL;
END Initialize_LCD;

Clear_display: PROCEDURE EXTERNAL;
END Clear_display;

LCD_print: PROCEDURE EXTERNAL;
END LCD_print;

DECLARE LCD_buffer          (48)  BYTE  PUBLIC,
       sign_on_message      (*)  BYTE  CONSTANT
       ('INTEL 8051 DRIVES LCD - '
        '2 ROWS BY 24 CHARACTERS '),
       1                     BYTE;

/* This is the start of the program */

/* Initialize the LCD */
CALL Initialize_LCD;
CALL Clear_display;

/* Now enter an endless loop to display the message */
DO WHILE 1;

  /* Copy the message to the buffer */
  DO i = 0 to 47;
    LCD_buffer(i) = sign_on_message(i);
  END;

  /* Now print out the buffer to the LCD */
  CALL LCD_print;

  /* wait a while */
  CALL Delay(2000);

  /* now clear the screen */
  CALL Clear_display;

END;          /* of DO WHILE */

END Main_module;

```

Main Module


```

LCD_IO_MODULE: DO;
DECLARE LCD_buffer (48) BYTE EXTERNAL,
LCD_command BYTE AT (08000H) AUXILIARY,
LCD_data BYTE AT (08001H) AUXILIARY,
LCD_status BYTE AT (08002H) AUXILIARY,
LCD_busy LITERALLY '1000$0000B',
1 BYTE;

Delay: PROCEDURE (msec) PUBLIC;

/* This procedure causes a delay of n msec */

DECLARE msec WORD,
1 WORD;

IF msec > 0 THEN DO;
DO i = 0 to msec - 1;
CALL Time(5); /* .2 msec delay */
END;

END Delay;

LCD_out: PROCEDURE (char) PUBLIC;

DECLARE char BYTE;

/* wait for LCD to indicate NOT busy */
DO WHILE (LCD_status AND LCD_busy) <> 0;
END;

/* now send the data to the LCD */
LCD_data = char;

END LCD out;

```

LCD Driver Module

LCD Driver Module (Continued)

```
LCD_command_out: PROCEDURE (char) PUBLIC;
```

```
DECLARE char BYTE;
```

```
/* wait for LCD to indicate NOT busy */
DO WHILE (LCD_status AND LCD_busy) <> 0;
END;
```

```
/* now send the command to the LCD */
LCD_command = char;
```

```
END LCD_command_out;
```

```
Home_cursor: PROCEDURE PUBLIC;
```

```
CALL LCD_command_out(0000$0010B);
```

```
END Home_cursor;
```

```
Clear_display: PROCEDURE PUBLIC;
```

```
CALL LCD_command_out (0000$0001B);
```

```
END Clear_display;
```

```
Set_cursor: PROCEDURE (position) PUBLIC;
```

```
DECLARE position
```

```
BYTE;
```

```
IF position > 47 THEN position = 47;
IF position < 24 THEN CALL LCD_command_out(080H + position);
ELSE CALL LCD_command_out(0C0H + (position - 24));
```

```
END Set_cursor;
```

```
Cursor_on: PROCEDURE PUBLIC;
```

```
CALL LCD_command_out(0000$1111B);
```

```
END Cursor_on;
```

```
Cursor_off: PROCEDURE PUBLIC;
```

```
CALL LCD_command_out(0000$1100B);
```

```
END Cursor_off;
```

LCD Driver Module (Continued)

```

LCD_print: PROCEDURE PUBLIC;

/* This procedure copies the contents of the LCD_buffer
   to the display */

CALL Set_cursor(0) ;
DO i = 0 to 23;
    CALL LCD_out(LCD_buffer(i));
END;
CALL Set_cursor(24);
DO i = 24 to 47;
    CALL LCD_out(LCD_buffer(i));
END;

END LCD_print;

```

```

Initialize_LCD: PROCEDURE PUBLIC;

CALL Delay(100);
CALL LCD_command_out(38H); /* Function Set */
CALL LCD_command_out(38H);
CALL LCD_command_out(06H); /* entry mode set */
CALL Clear_display;
CALL Home_cursor;
CALL Cursor_off;
CALL Set_cursor(0);

END Initialize_LCD;

END LCD_IO_Module;

```

LCD Driver Module (Continued)

RICK SCHUE
 REGIONAL APPLICATIONS SPECIALIST
 INDIANAPOLIS, INDIANA



APPLICATION BRIEF

AB-40

```
END:
CALL Set_cursor(0);
DO i = 0 to 255;
CALL LCD_out(LCD_buffer(i));
END;
CALL Set_cursor(24);
DO i = 24 to 47;
CALL LCD_out(LCD_buffer(i));
END;
```

December 1987

```
InitializeLCD: PROCEDURE PUBLIC;
CALL Delay(100);
CALL LCD_command_out(3BH); /* Function Set */
CALL LCD_command_out(3BH);
CALL LCD_command_out(06H); /* entry mode set */
CALL Clear_display;
CALL Home_cursor;
CALL Cursor_off;
CALL Set_cursor(0);
```

32-Bit Math Routines for the 8051

LCD Driver Module (Continued)

RICK SCHUE
REGIONAL APPLICATIONS SPECIALIST
INDIANAPOLIS, INDIANA

Here are some easy to use 16- and 32-bit math routines that take the pain out of calculations such as PID loops, A/D calibration, linearization calculations and anything else that requires 32-bit accuracy.

The package is written to interface with PL/M-51. Parameters are passed as 16-bit words to the routines, which perform operations on a 32-bit "accumulator" resident in memory. The following functions are performed:

Load_16 (word_param)

Loads a 16-bit -RD into the low half of the 32-bit "accumulator", zeros upper 16 bits of accumulator.

Load_32 (word_hi,word_lo)

Loads word_hi into upper 16 bits of accumulator, word_lo into Lower 16 bits.

Low_16

Returns the lower 16 bits of the accumulator, bits 0 through 15.

Mid_16

Returns the middle 16 bits of the accumulator, bits 8 through 23.

High_16

Returns the upper 16 bits of the accumulator, bits 16 through 31.

Mul_16 (word_param)

Multiplies the 32-bit accumulator by the 16-bit word supplied, result left in accumulator.

Div_16 (word_param)

Divides the 32-bit accumulator by the 16-bit word supplied, result left in accumulator.

Add_16 (word_param)

Adds the 16-bit word supplied to the 32-bit accumulator.

Sub_16 (word_param)

Similar to Add_16 but for subtraction.

Add_32 (word_hi,word_lo)

Forms a 32-bit value for word_hi and word_lo and adds it to the accumulator.

Sub_32 (word_hi,word_lo)

Similar to Add_32 but for subtraction.

APPLICATION

Typical applications have 16-bit "input" values and produce 16-bit "output" values, but require 32-bit values for intermediate results. An example would be reading a 12-bit A/D, performing some gain and offset calculation on the raw A/D data to produce a calibrated 16 bit result. Doing this is a simple task with this math package.

```
CALL Load_16(AD_value);
```

```
CALL Add_16 (offset_value);
```

```
CALL Mul_16 (gain_factor);
```

```
/* gain is in units of 1/256 */
```

```
result = Mid_16;
```

In this example the accumulator was loaded with the raw A/D value and then the offset was applied. The gain_factor was "pre-multiplied" by 256 (8 bits), giving it a granularity of 1/256. The result was extracted from the "middle" 16 bits of the accumulator (bits 8 to 23) to account for the scaling factor of 256 introduced in the multiply step.

The package requires about 384 bytes of ROM and 30 bytes of RAM. Individual routines can be deleted to conserve RAM if they are not used.

CODE SOURCE LISTINGS

CODE SOURCE LISTINGS

```

NAME      Math_32_Module
;
PUBLIC    Load_16, ?Load_16?byte
PUBLIC    Load_32, ?Load_32?byte
PUBLIC    Mul_16, ?Mul_16?byte
PUBLIC    Div_16, ?Div_16?byte
PUBLIC    Add_16, ?Add_16?byte
PUBLIC    Sub_16, ?Sub_16?byte
PUBLIC    Add_32, ?Add_32?byte
PUBLIC    Sub_32, ?Sub_32?byte
PUBLIC    Low_16, Mid_16, High_16
;
Math_32_Data SEGMENT DATA
Math_32_Code SEGMENT CODE
;
RSEG      Math_32_Data
?Load_16?byte: DS 2
?Load_32?byte: DS 4
?Mul_16?byte: DS 2
?Div_16?byte: DS 2
?Add_16?byte: DS 2
?Sub_16?byte: DS 2
?Add_32?byte: DS 4
?Sub_32?byte: DS 4
OP_0: DS 1
OP_1: DS 1
OP_2: DS 1
OP_3: DS 1
TMP_0: DS 1
TMP_1: DS 1
TMP_2: DS 1
TMP_3: DS 1
;
RSEG      Math_32_Code

```

The package requires about 384 bytes of ROM and 30 bytes of RAM. Individual routines can be deleted to conserve RAM if they are not used.

There are some easy to use 16- and 32-bit math routines. The package is written to interface with PLM-M-2. Parameters are passed as 16-bit words to the routines, which perform operations on a 32-bit "accumulator" resident in memory. The following functions are performed:

Load_16 (word_param)
Loads a 16-bit R16 into the low half of the 32-bit "accumulator", zeros upper 16 bits of accumulator.

Load_32 (word_param)
Loads word into upper 16 bits of accumulator, word into lower 16 bits.

Low_16
Returns the lower 16 bits of the accumulator, bits 0 through 15.

Mid_16
Returns the middle 16 bits of the accumulator, bits 8 through 23.

High_16
Returns the upper 16 bits of the accumulator, bits 16 through 31.

Mul_16 (word_param)
Multiplies the 32-bit accumulator by the 16-bit word 270530-1

Div_16 (word_param)
Divides the 32-bit accumulator by the 16-bit word supplied, result left in accumulator.

Add_16 (word_param)
Adds the 16-bit word supplied to the 32-bit accumulator.

Sub_16 (word_param)
Subtracts the 16-bit word supplied from the 32-bit accumulator.

```

Load_16:
;Load the lower 16 bits of the OP registers with the value supplied
MOV     OP_3,#0
MOV     OP_2,#0
MOV     OP_1,?Load_16?byte
MOV     OP_0,?Load_16?byte + 1
RET

Load_32:
;Load all the OP registers with the value supplied
MOV     OP_3,?Load_32?byte
MOV     OP_2,?Load_32?byte + 1
MOV     OP_1,?Load_32?byte + 2
MOV     OP_0,?Load_32?byte + 3
RET

Low_16:
;Return the lower 16 bits of the OP registers
MOV     R6,OP_1
MOV     R7,OP_0
RET

Mid_16:
;Return the middle 16 bits of the OP registers
MOV     R6,OP_2
MOV     R7,OP_1
RET

High_16:
;Return the high 16 bits of the OP registers
MOV     R6,OP_3
MOV     R7,OP_2
RET

Add_16:
;Add the 16 bits supplied by the caller to the OP registers
CLR     C
MOV     A,OP_0
ADDC    A,?Add_16?byte + 1 ;low byte first
MOV     OP_0,A
MOV     A,OP_1
ADDC    A,?Add_16?byte ;high byte + carry
MOV     OP_1,A
MOV     A,#0
ADDC    A,#0 ;propagate carry only
MOV     OP_2,A
MOV     A,OP_3
ADDC    A,#0 ;propagate carry only
MOV     OP_3,A
RET

```

270530-2

```

Add_32:
;Add the 32 bits supplied by the caller to the OP registers
CLR C
MOV A,OP_0
ADDC A,?Add_32?byte + 3 ;lowest byte first
MOV OP_0,A
MOV A,OP_1
ADDC A,?Add_32?byte + 2 ;mid-lowest byte + carry
MOV OP_1,A
MOV A,OP_2
ADDC A,?Add_32?byte + 1 ;mid-highest byte + carry
MOV OP_2,A
MOV A,OP_3
ADDC A,?Add_32?byte ;highest byte + carry
MOV OP_3,A
RET

Sub_16:
;Subtract the 16 bits supplied by the caller from the OP registers
CLR C
MOV A,OP_0
SUBB A,?Sub_16?byte + 1 ;low byte first
MOV OP_0,A
MOV A,OP_1
SUBB A,?Sub_16?byte ;high byte + carry
MOV OP_1,A
MOV A,OP_2
SUBB A,#0 ;propagate carry only
MOV OP_2,A
MOV A,OP_3
SUBB A,#0 ;propagate carry only
MOV OP_3,A
RET

Sub_32:
;Subtract the 32 bits supplied by the caller from the OP registers
CLR C
MOV A,OP_0
SUBB A,?Sub_32?byte + 3 ;lowest byte first
MOV OP_0,A
MOV A,OP_1
SUBB A,?Sub_32?byte + 2 ;mid-lowest byte + carry
MOV OP_1,A
MOV A,OP_2
SUBB A,?Sub_32?byte + 1 ;mid-highest byte + carry
MOV OP_2,A
MOV A,OP_3
SUBB A,?Sub_32?byte ;highest byte + carry
MOV OP_3,A
RET

```

270530-3


```

Mul_16:
; Multiply the 32 bit OP with the 16 value supplied
MOV     TMP_3,#0           ; clear out upper 16 bits
MOV     TMP_2,#0           ; save
; Generate the lowest byte of the result
MOV     B,OP_0
MOV     A,?Mul_16?byte+1
MUL     AB
MOV     TMP_0,A           ; low-order result
MOV     TMP_1,B           ; high order result
; Now generate the next higher order byte
MOV     B,OP_1
MOV     A,?Mul_16?byte+1
MUL     AB
ADD     A,TMP_1           ; low-order result
MOV     TMP_1,A           ; save
MOV     A,B               ; get high-order result
ADDC    A,TMP_2           ; include carry from previous operation
MOV     TMP_2,A           ; save
JNC     Mul_loop1
INC     TMP_3             ; propagate carry into TMP_3

Mul_loop1:
MOV     B,OP_0
MOV     A,?Mul_16?byte
MUL     AB
ADD     A,TMP_1           ; low-order result
MOV     TMP_1,A           ; save
MOV     A,B               ; get high-order result
ADDC    A,TMP_2           ; include carry from previous operation
MOV     TMP_2,A           ; save
JNC     Mul_loop2
INC     TMP_3             ; propagate carry into TMP_3

Mul_loop2:
; Now start working on the 3rd byte
MOV     B,OP_2
MOV     A,?Mul_16?byte+1
MUL     AB
ADD     A,TMP_2           ; low-order result
MOV     TMP_2,A           ; save
MOV     A,B               ; get high-order result
ADDC    A,TMP_3           ; include carry from previous operation
MOV     TMP_3,A           ; save
; Now the other half
MOV     B,OP_1
MOV     A,?Mul_16?byte
MUL     AB
ADD     A,TMP_2           ; low-order result
MOV     TMP_2,A           ; save
MOV     A,B               ; get high-order result
ADDC    A,TMP_3           ; include carry from previous operation
MOV     TMP_3,A           ; save
; Now finish off the highest order byte
MOV     B,OP_3
MOV     A,?Mul_16?byte+1

```

270530-4

```

MUL     AB
ADD     A,TMP_3           ;low-order result
MOV     TMP_3,A           ; save
; Forget about the high-order result, this is only 32 bit math!
MOV     B,OP_2
MOV     A,?Mul_16?byte
MUL     AB
ADD     A,TMP_3           ;low-order result
MOV     TMP_3,A           ; save
; Now we are all done, move the TMP values back into OP
MOV     OP_0,TMP_0
MOV     OP_1,TMP_1
MOV     OP_2,TMP_2
MOV     OP_3,TMP_3
RET

```

```

Div_16:
;This divides the 32 bit OP register by the value supplied
MOV R7,#0
MOV R6,#0 ;zero out partial remainder
MOV TMP_0,#0
MOV TMP_1,#0
MOV TMP_2,#0
MOV TMP_3,#0
MOV R1,?Div_16?byte ;load divisor
MOV R0,?Div_16?byte+1
MOV R5,#32 ;loop count
;This begins the loop
Div_loop:
CALL Shift_D ;shift the dividend and return MSB in C
MOV A,R6 ;shift carry into LSB of partial remainder
RLC A
MOV R6,A
MOV A,R7
RLC A
MOV R7,A
;now test to see if R7:R6 >= R1:R0
CLR C
MOV A,R7 ;subtract R1 from R7 to see if R1 < R7
SUBB A,R1 ; A = R7 - R1, carry set if R7 < R1
JC Cant_sub
;at this point R7>R1 or R7=R1
JNZ Can_sub ;jump if R7>R1
;if R7 = R1, test for R6>=R0
CLR C
MOV A,R6
SUBB A,R0 ; A = R6 - R0, carry set if R6 < R0
JC Cant_sub
Can_sub:
;subtract the divisor from the partial remainder
CLR C
MOV A,R6
SUBB A,R0 ; A = R6 - R0
MOV R6,A
MOV A,R7
SUBB A,R1 ; A = R7 - R1 - Borrow
MOV R7,A
SETB C ; shift a 1 into the quotient
JMP Quot
Cant_sub:
;shift a 0 into the quotient
CLR C
Quot:
;shift the carry bit into the quotient
CALL Shift_Q
; Test for completion
DJNZ R5,Div_loop
; Now we are all done, move the TMP values back into OP
MOV OP_0,TMP_0
MOV OP_1,TMP_1

```

270530-6

```

MOV     OP_2,TMP_2
MOV     OP_3,TMP_3
RET

```

```

Shift_D:
;shift the dividend one bit to the left and return the MSB in C

```

```

CLR     C
MOV     A,OP_0
RLC     A
MOV     OP_1,A
MOV     A,OP_1
RLC     A
MOV     OP_2,A
MOV     A,OP_2
RLC     A
MOV     OP_3,A
RET

```

```

Shift_Q:
;shift the quotient one bit to the left and shift the C into LSB

```

```

MOV     A,TMP_0
RLC     A
MOV     TMP_0,A
MOV     A,TMP_1
RLC     A
MOV     TMP_1,A
MOV     A,TMP_2
RLC     A
MOV     TMP_2,A
MOV     A,TMP_3
RLC     A
MOV     TMP_3,A
RET

```

END

```

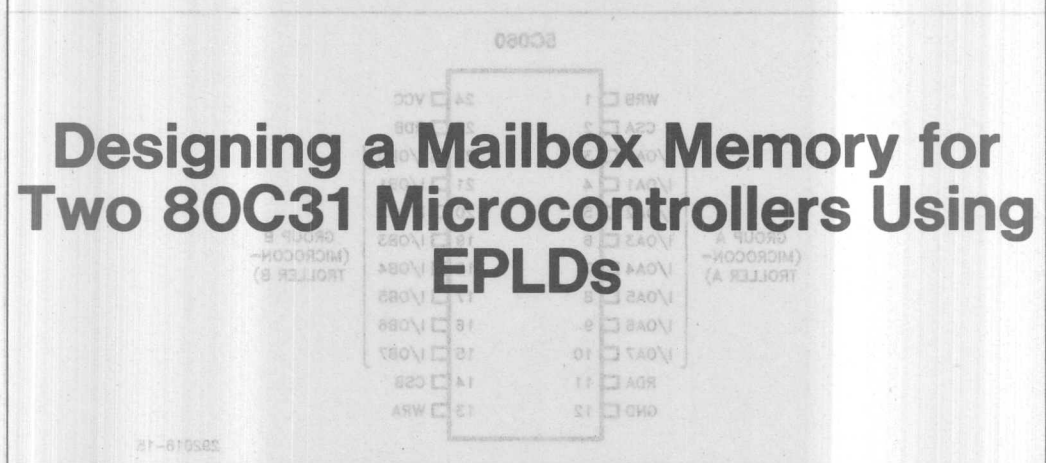
;this divides the 32 bit OP register by the value supplied
;zero out partial remainder
MOV     R3,#0
MOV     TMP_0,#0
MOV     TMP_1,#0
MOV     TMP_2,#0
MOV     TMP_3,#0
MOV     R1,TMP_16byte
MOV     R0,TMP_16byte+1
MOV     R2,#32
;this begins the loop
Div_Loop:
CALL    Shift_D
;shift the dividend and return MSB in C
;shift carry into LSB of partial remainder
MOV     A,R0
RLC     A
MOV     R0,A
MOV     A,R1
RLC     A
MOV     R1,A
;now test to see if R0:R2 >= R1:R0
CLR     C
MOV     A,R0
SUBB    A,R1
JC       Carry_Sup
;set this point R0:R1 or R1:R1
;jump if R0:R1
MOV     A,R1
RLC     A
MOV     TMP_1,A
CLR     C
MOV     A,R0
SUBB    A,R1
JC       Carry_Sup
;A = R0 - R1, carry set if R0 < R1
;subtract the divisor from the partial remainder
CLR     C
MOV     A,R0
SUBB    A,R1
MOV     R0,A
;A = R0 - R1
;shift a 0 into the quotient
CLR     C
;shift the carry bit into the quotient
CALL    Shift_Q
;Test for completion
JNZ     R2,Div_Loop
;Now we are all done, move the TMP values back into OP
MOV     OP_0,TMP_0
MOV     OP_1,TMP_1

```

270530-7

270530-8

October 1987



K. WEIGL & J. STAHL
INTEL CORPORATION
MUNICH, GERMANY

INTRODUCTION

Very often, complex systems involve two or more microcontrollers to fulfill the requirements defined by a given objective. Since the nature of microcontrollers does not allow for easy dual-port memory design (no "READY" input; no "HOLD/HLDA" interface; port-oriented I/O etc.), design engineers are faced with the problem of interchanging information (data and status) between those microcontrollers. This application brief describes the design of a mailbox for exchanging information between two 80C31s, using a 5C060 H-EPLD as a "back-to-back" register, and a 5C031 H-EPLD as an arbitration vehicle to control the actions of the CPUs.

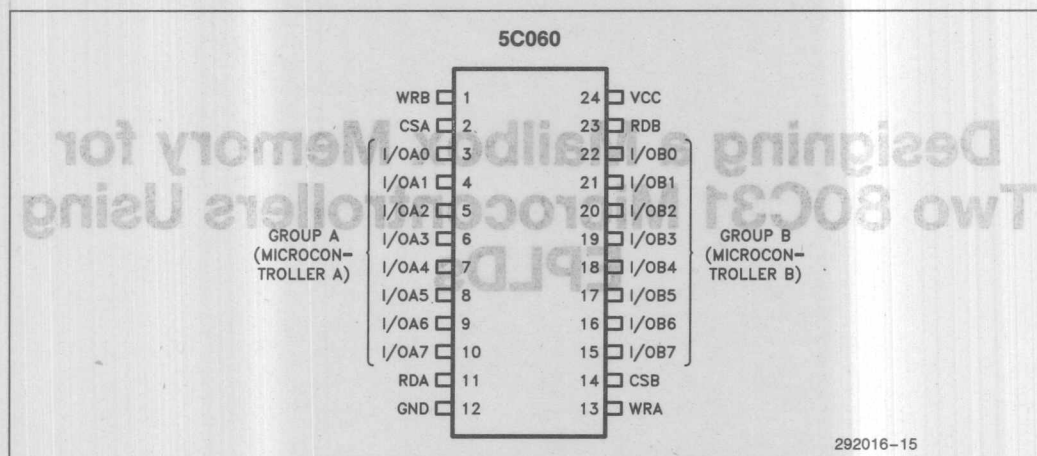
October 1987

THE 5C060 MAILBOX

In this application, the 16 macrocells of the 5C060 are grouped into two sets of 8 so called "ROIF" (register output with input feedback) primitives to implement the two 8 bit bus interfaces needed. The grouping is done according to the following picture.

The 5C060 allows for independent clocking of 8 macrocells on each side of the chip, the two clock inputs are used to clock data from the microcontroller bus into the chip. To read the data written into the mailbox by one of the controllers, the RDA- (controller A is reading) or RDB- (controller B is reading) line must be pulled low by activating the read command (/RD). In order to avoid spurious read-cycles, the /RD commands from both microcontrollers are logically "ORed" together with an active high CS-signal (Chip Select) inside the 5C060. The CS-signal for both ports is derived from address line A15. Therefore, whenever A15 becomes a logic "1" (true), the mailbox is activated and ready to take or submit data.

Address range for the mailbox: F000 Hex to FFFF Hex
(Upper 12 kbyte)



K. WEIGL & J. STAHL
INTEL CORPORATION
MUNICH, GERMANY

THE 5C031 "MAILBOX CONTROLLER"

To keep the two microcontrollers informed about the status of their mailbox, the 5C031 is programmed to supply the following signals to both controllers:

/OBFA: "OUTPUT BUFFER FULL" FOR MC A

/OBFB: "OUTPUT BUFFER FULL" FOR MC B

/IBEA: "INPUT BUFFER EMPTY" FOR MC A

/IBEB: "INPUT BUFFER EMPTY" FOR MC B

/INTA: INTERRUPT TO MC A

/INTB: INTERRUPT TO MC B

The next section will discuss the meanings of these signals in more detail.

Output Buffer Full: This flag is set whenever the controller writes into its own output buffer. The flag remains valid, until the second controller has read the data. The flag is automatically reset to its inactive state when this read cycle is accomplished.

NOTE:

Both controllers can access (read or write) the mailbox simultaneously.

Input Buffer Empty: This flag indicates that there is no message in the mailbox. The flag will become inactive as soon as one microcontroller places a message for the other one (or vice versa).

Example: /IBEA remains "LOW" until microcontroller B places a message for controller A. /IBEA will go "HIGH" as soon as controller B has accomplished its write cycle, and will not go "LOW" again until microcontroller A has read the message.

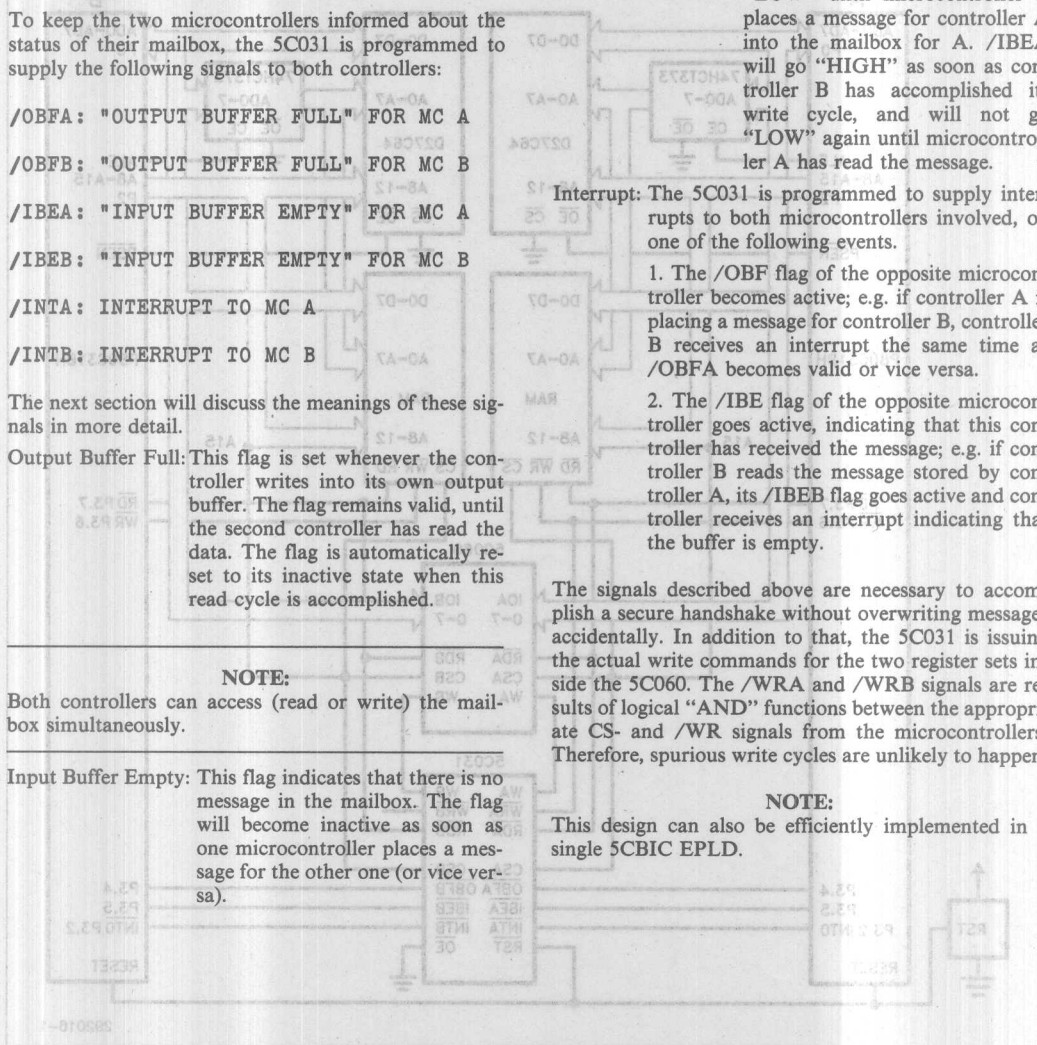
Interrupt: The 5C031 is programmed to supply interrupts to both microcontrollers involved, on one of the following events.

1. The /OBF flag of the opposite microcontroller becomes active; e.g. if controller A is placing a message for controller B, controller B receives an interrupt the same time as /OBFA becomes valid or vice versa.
2. The /IBE flag of the opposite microcontroller goes active, indicating that this controller has received the message; e.g. if controller B reads the message stored by controller A, its /IBEB flag goes active and controller receives an interrupt indicating that the buffer is empty.

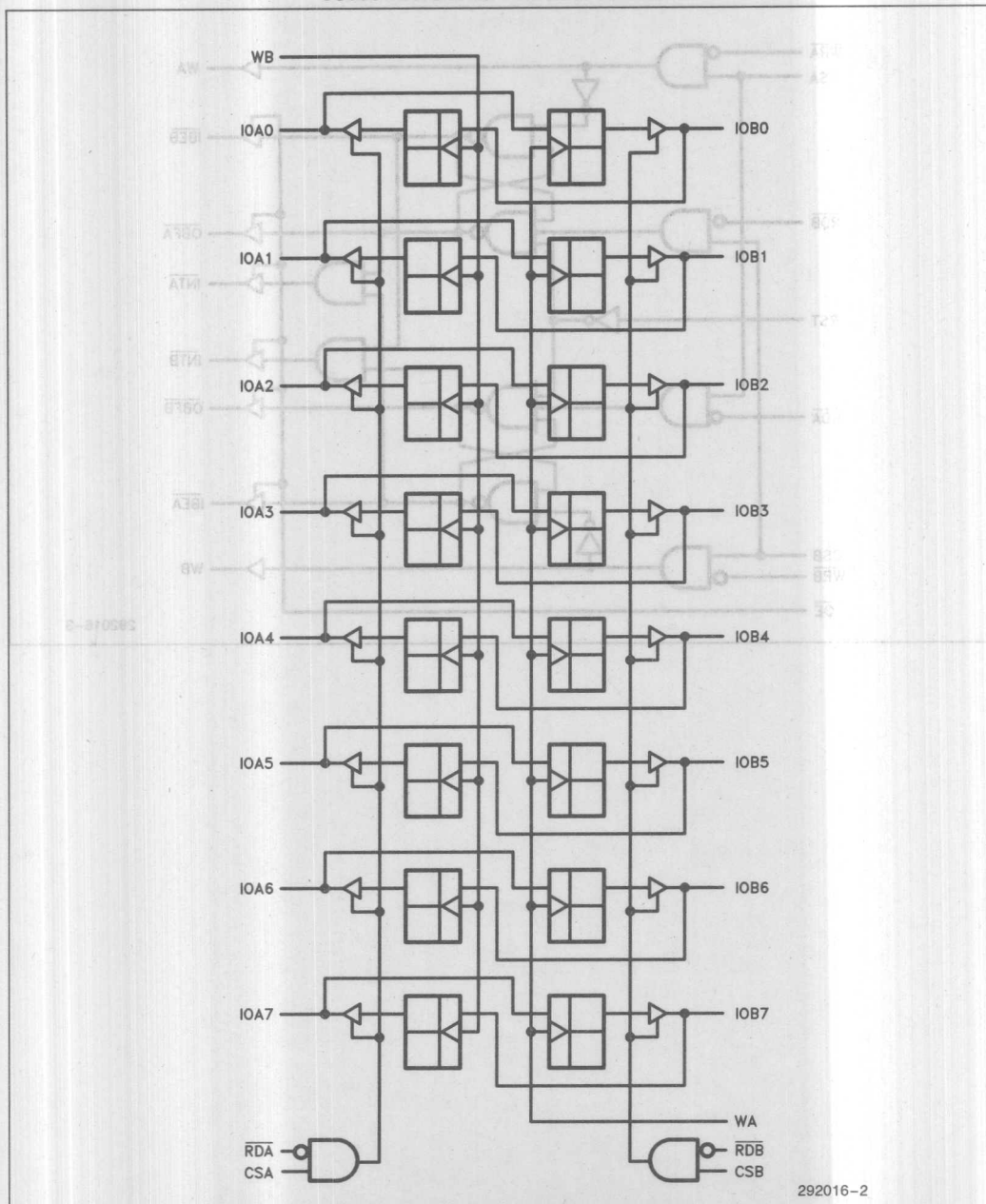
The signals described above are necessary to accomplish a secure handshake without overwriting messages accidentally. In addition to that, the 5C031 is issuing the actual write commands for the two register sets inside the 5C060. The /WRA and /WRB signals are results of logical "AND" functions between the appropriate CS- and /WR signals from the microcontrollers. Therefore, spurious write cycles are unlikely to happen.

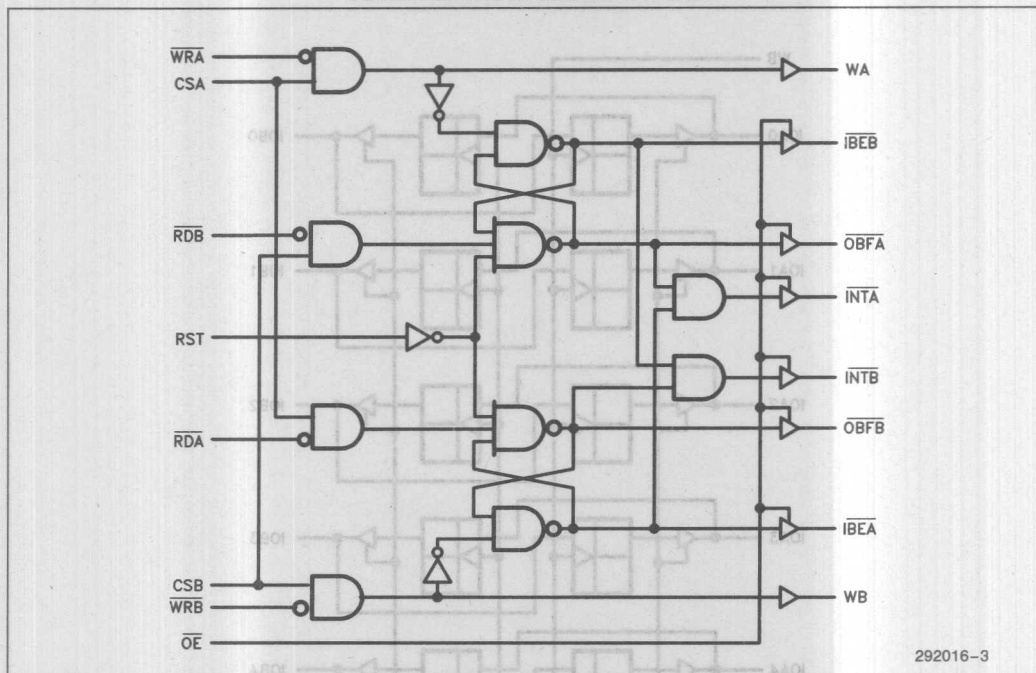
NOTE:

This design can also be efficiently implemented in a single 5CBIC EPLD.



5C060 "BACK TO BACK REGISTER"





5C060 REGISTER ADF

```

JUERG STAHL
INTEL ZUERICH
March 27, 1986
80C31 MAILBOX MEMORY USING 5C060 / 5C031
1
5C060

LB Version 3.0, Baseline 17x, 9/26/85
PART: 5C060
INPUTS: WB@1, CSA@2, CSB@14, nRDA@11, nRDB@23, WA@13
OUTPUTS: IOB7@15, IOA7@10, IOB6@16, IOA6@9,
         IOB5@17, IOA5@8, IOB4@18, IOA4@7,
         IOB3@19, IOA3@6, IOB2@20, IOA2@5,
         IOB1@21, IOA1@4, IOB0@22, IOA0@3

NETWORK:
IOB7,DB7 = ROIF (DA7,WAC,GND,GND,RDBC)
IOA7,DA7 = ROIF (DB7,WBC,GND,GND,RDAC)
IOB6,DB6 = ROIF (DA6,WAC,GND,GND,RDBC)
IOA6,DA6 = ROIF (DB6,WBC,GND,GND,RDAC)
IOB5,DB5 = ROIF (DA5,WAC,GND,GND,RDBC)
IOA5,DA5 = ROIF (DB5,WBC,GND,GND,RDAC)
IOB4,DB4 = ROIF (DA4,WAC,GND,GND,RDBC)
IOA4,DA4 = ROIF (DB4,WBC,GND,GND,RDAC)
IOB3,DB3 = ROIF (DA3,WAC,GND,GND,RDBC)
IOA3,DA3 = ROIF (DB3,WBC,GND,GND,RDAC)
IOB2,DB2 = ROIF (DA2,WAC,GND,GND,RDBC)
IOA2,DA2 = ROIF (DB2,WBC,GND,GND,RDAC)
IOB1,DB1 = ROIF (DA1,WAC,GND,GND,RDBC)
IOA1,DA1 = ROIF (DB1,WBC,GND,GND,RDAC)
IOB0,DB0 = ROIF (DA0,WAC,GND,GND,RDBC)
IOA0,DA0 = ROIF (DB0,WBC,GND,GND,RDAC)
WAC = INP (WA)
RDBC = AND(CSBI,RDBI)
WBC = INP (WB)
RDAC = AND(CSAI,RDAI)
CSBI = INP (CSB)
nRDBI = INP(nRDB)
nRDAI = INP(nRDA)
CSAI = INP(CSA)
RDAI = NOT(nRDAI)
RDBI = NOT(nRDBI)

END$

```

292016-4

5C060 REGISTER LEF

```

JUEBG STAHL
INTEL ZUERICH
March 27, 1986
80C31 MAILBOX MEMORY USING 5C060 / 5C031
1
5C060

LB Version 3.0, Baseline 17x, 9/26/85
LEF Version 1.0 Baseline 1.51 02 Feb 1987
PART:
5C060

INPUTS:
WB@1, CSA@2, CSB@14, nRDA@11, nRDB@23, WA@13

OUTPUTS:
IOB7@15, IOA7@10, IOB6@16, IOA6@9, IOB5@17, IOA5@8, IOB4@18, IOA4@7,
IOB3@19, IOA3@6, IOB2@20, IOA2@5, IOB1@21, IOA1@4, IOB0@22, IOA0@3

NETWORK:
WBC = INP(WB)
WAC = INP(WA)
CSAI = INP(CSA)
CSBI = INP(CSB)
nRDAI = INP(nRDA)
nRDBI = INP(nRDB)
IOB7, DB7 = ROIF(DA7, WAC, GND, GND, RDBC)
IOA7, DA7 = ROIF(DB7, WBC, GND, GND, RDAC)
IOB6, DB6 = ROIF(DA6, WAC, GND, GND, RDBC)
IOA6, DA6 = ROIF(DB6, WBC, GND, GND, RDAC)
IOB5, DB5 = ROIF(DA5, WAC, GND, GND, RDBC)
IOA5, DA5 = ROIF(DB5, WBC, GND, GND, RDAC)
IOB4, DB4 = ROIF(DA4, WAC, GND, GND, RDBC)
IOA4, DA4 = ROIF(DB4, WBC, GND, GND, RDAC)
IOB3, DB3 = ROIF(DA3, WAC, GND, GND, RDBC)
IOA3, DA3 = ROIF(DB3, WBC, GND, GND, RDAC)
IOB2, DB2 = ROIF(DA2, WAC, GND, GND, RDBC)
IOA2, DA2 = ROIF(DB2, WBC, GND, GND, RDAC)
IOB1, DB1 = ROIF(DA1, WAC, GND, GND, RDBC)
IOA1, DA1 = ROIF(DB1, WBC, GND, GND, RDAC)
IOB0, DB0 = ROIF(DA0, WAC, GND, GND, RDBC)
IOA0, DA0 = ROIF(DB0, WBC, GND, GND, RDAC)

EQUATIONS:
RDAC = CSAI * nRDAI';
RDBC = CSBI * nRDBI';

END$

```

292016-5

(hour) 5C060 REGISTER UTILIZATION REPORT 0003

Logic Optimizing Compiler Utilization Report
FIT Version 1.0 Baseline 1.0i 2/6/87

***** Design implemented successfully

JUERG STAHL
INTEL ZUERICH
March 27, 1986
80C31 MAILBOX MEMORY USING 5C060 / 5C031
1
5C060

** EXAMPLE .RPT FILE **

LB Version 3.0, Baseline 17x, 9/26/85

5C060

WB --: 1 24:- Vcc
CSA --: 2 23:- nRDB
IOA0 --: 3 22:- IOB0
IOA1 --: 4 21:- IOB1
IOA2 --: 5 20:- IOB2
IOA3 --: 6 19:- IOB3
IOA4 --: 7 18:- IOB4
IOA5 --: 8 17:- IOB5
IOA6 --: 9 16:- IOB6
IOA7 --:10 15:- IOB7
nRDA --:11 14:- CSB
GND --:12 13:- WA

INPUTS

Name	Pin	Resource	MCell #	PTerms	MCells	OE	Clear	Clock
WB	1	INP	1	8 \1	01	-	-	CLK1
CSA	2	INP	2	8 \1	11	9	-	-
			3	8 \1	21	10	-	-
			4	8 \1	31	11	-	-
			5	8 \1	41	12	-	-
			6	8 \1	51	13	-	-
			7	8 \1	61	14	-	-
			8	8 \1	71	15	-	-
nRDA	11	INP	9	8 \1	81	16	-	-
			10	8 \1	91	9	-	-
			11	8 \1	01	10	-	-
			12	8 \1	11	11	-	-
			13	8 \1	21	12	-	-
			14	8 \1	31	13	-	-
			15	8 \1	41	14	-	-
			16	8 \1	51	15	-	-
GND	12	GND	17	8 \1	61	16	-	-
			18	8 \1	71	1	-	-
			19	8 \1	81	2	-	-
			20	8 \1	91	3	-	-
			21	8 \1	01	4	-	-
			22	8 \1	11	5	-	-
			23	8 \1	21	6	-	-
			24	8 \1	31	7	-	-
			25	8 \1	41	8	-	-
			26	8 \1	51	9	-	-

292016-6

Device	Pin	Signal	Value	Unit	Notes
WA	13	INP	0	V	Input pin 13
CSB	14	INP	0	V	Input pin 14
nRDB	23	INP	0	V	Input pin 23
Vcc	24	Vcc	1.5	V	Power pin 24

Name	Pin	Resource	MCell #	PTerms	MCells	OE	Clear	Clock
IOA0	3	ROIF	9	1/ 8	1	-	-	-
IOA1	4	ROIF	10	1/ 8	2	-	-	-
IOA2	5	ROIF	11	1/ 8	3	-	-	-
IOA3	6	ROIF	12	1/ 8	4	-	-	-
IOA4	7	ROIF	13	1/ 8	5	-	-	-
IOA5	8	ROIF	14	1/ 8	6	-	-	-
IOA6	9	ROIF	15	1/ 8	7	-	-	-
IOA7	10	ROIF	16	1/ 8	8	-	-	-
IOB7	15	ROIF	8	1/ 8	16	-	-	-
IOB6	16	ROIF	7	1/ 8	15	-	-	-
IOB5	17	ROIF	6	1/ 8	14	-	-	-
IOB4	18	ROIF	5	1/ 8	13	-	-	-
IOB3	19	ROIF	4	1/ 8	12	-	-	-
IOB2	20	ROIF	3	1/ 8	11	-	-	-
IOB1	21	ROIF	2	1/ 8	10	-	-	-
IOB0	22	ROIF	1	1/ 8	9	-	-	-

292016-8

2-186

5C031 ARBITER LEF (Continued)

Logic Optimizing Compiler Utilization Report
FIT Version 1.0 Baseline 1.0i 2/6/87

**** Design implemented successfully

JUERG STAHL
INTEL ZUERICH
March 28, 1986
80C31 MAILBOX MEMORY USING 5C060 / 5C031
2
5C031

LB Version 3.0, Baseline 17x, 9/26/85

5C031
GND --: 1 20:- Vcc
GND --: 2 19:- WB
nOE --: 3 18:- WA
CSB --: 4 17:- nOBFb
nWRB --: 5 16:- nINTB
nRDA --: 6 15:- nINTA
CSA --: 7 14:- nIBEB
nRDB --: 8 13:- nOBFA
nWRA --: 9 12:- nIBEA
GND --:10 11:- RST

INPUTS

Name	Pin	Resource	MCell #	PTerms	MCells	Feeds:	OE	Clear	Preset
nOE	3	INP	-	-	-	3 4 5 6 7 8			
CSB	4	INP	-	-	1 7 8				
nWRB	5	INP	-	-	1 8				
nRDA	6	INP	-	-	3				
CSA	7	INP	-	-	2 3 6				
nRDB	8	INP	-	-	7				
nWRA	9	INP	-	-	2 6				
GND	10	GND	-	-	-				
RST	11	INP	-	-	3 7				
Vcc	20	Vcc	-	-	-	1 2			

292016-11

****OUTPUTS****

Name	Pin	Resource	MCell #	PTerms	Feeds:	MCells	OK	Clear	Preset
nIBEA	12	COCF	8	2/ 8	3	-	-	-	-
nOBFA	13	COCF	7	2/ 8	5	-	-	-	-
nIBEB	14	COCF	6	2/ 8	4	-	-	-	-
nINTA	15	CONF	5	1/ 8	-	-	-	-	-
nINTB	16	CONF	4	1/ 8	-	-	-	-	-
nOBFB	17	COCF	3	2/ 8	4	-	-	-	-
WA	18	CONF	2	1/ 8	-	-	-	-	-
WB	19	CONF	1	1/ 8	-	-	-	-	-

****UNUSED RESOURCES****

Name	Pin	Resource	MCell	PTerms
-	1	-	-	-
-	2	-	-	-

****PART UTILIZATION****

88%	Pins
100%	MacroCells
18%	PTerms

292016-12

NOTE

Both processors have their advantages and disadvantages, which are largely transparent to the user.

Lower operating voltages are easier to obtain with the p-well structure than with the n-well structure. But the p-well structure does not easily adapt to an EPROM which would be pin-for-pin compatible with HMOS EPROMs. On the other hand the n-well structure can be based on the solidly founded HMOS process in which PMOS and NMOS are built into a p-type substrate. This allows somewhat more than half of the transistor in a CMOS chip to be constructed by processes that are already well characterized.

Currently Intel's CMOS microcontrollers and microprocessors are p-well devices.

Further discussion of the CMOS technology is provided in References 1 and 2 (which are reprinted in the Microcontroller Handbook).

THE MCS-81 FAMILY IN CMOS

The 80C51BH is the CMOS version of Intel's original 8051. The 80C51BH is the ROMless 80C51BH, relative to the 8051. These CMOS devices are microprocessor-compatible with their HMOS counterparts except for power consumption and speed.

In most cases, an 80C51BH can directly replace the 8051 in existing applications. It can execute the same code at the same speed, accept signals from the same sources, and drive the same loads. However, the 80C51BH covers a wider range of speeds, will emit CMOS logic levels to CMOS loads, and will draw about 1/10 the current of an 8051 (and less yet in the reduced power modes). Interchangeability between the HMOS and CMOS devices is discussed in more detail in the final section of this Application Note.

It should be noted that the 80C51BH CPU is not static. That means if the clock frequency is too low, the CPU might forget what it was doing. This is because the circuitry uses a number of dynamic nodes. A dynamic node is one that uses the not-to-ground capacitance to form a temporary storage cell. Dynamic nodes are used to reduce the transistor count, and hence the chip area, and hence the cost of the device.

It is not to say that the on-chip RAM in CMOS microcontrollers is dynamic. It's not. It's the CPU that is dynamic, and that is what induces the minimum clock frequency specification.

CMOS EVOLVES

The original CMOS logic families were the 4000-series and the 74C-series circuits. The 74C-series circuits are additional equivalents to the corresponding numbered 74-series TTL circuits, but have CMOS logic levels and retain the other well-known characteristics of CMOS logic.

These characteristics are: low power consumption, high noise immunity, and slow speed. The low power consumption is inherent to the nature of the CMOS circuit. The noise immunity is due partly to the CMOS logic levels, and partly to the slowness of the circuits. The slow speed is due to the technology used to construct the circuit.

March 1985

The technology used is called metal-gate CMOS, because the metal gates are formed by metal deposition. After the gates are formed after the silicon, metal impurities have been defined, and must overlap the source and drain somewhat to allow for alignment tolerances. This overlap plus the relatively large size of the transistors themselves result in high parasitic capacitance, and that is what limits the speed of the circuit.

High speed CMOS became feasible with the development of the silicon gate technology. In this process, the gates are formed by an implantation of polysilicon into the silicon. This eliminates the need for the overlap capacitance. In addition, the process allows smaller transistors. The result is a significant increase in circuit speed. The 74HC-series of CMOS logic circuits is based on this technology, and has speeds comparable to 1.5 TTL, which is to say about 10 times faster than the 74C-series circuits.

The speed reduction that contributes to the higher speed also demands a corresponding reduction in the maximum supply voltage. High-speed CMOS is generally limited to 5V.

WHAT IS CMOS?

CMOS is the name given to Intel's high-speed CMOS processes. There are two CMOS processes, one based on an n-well structure and one based on a p-well structure. In the n-well structure, n-type wells are diffused into a p-type substrate. Then the n-channel PMOS (PMT) are built into the substrate. In the p-well structure, p-type wells are diffused into an n-type substrate. Then the PMT are built into the wells and PMT into the

TOM WILLIAMSON
MCO APPLICATIONS ENGINEER

CMOS EVOLVES

The original CMOS logic families were the 4000-series and the 74C-series circuits. The 74C-series circuits are functional equivalents to the corresponding numbered 74-series TTL circuits, but have CMOS logic levels and retain the other well known characteristics of CMOS logic.

These characteristics are: low power consumption, high noise immunity, and slow speed. The low power consumption is inherent to the nature of the CMOS circuit. The noise immunity is due partly to the CMOS logic levels, and partly to the slowness of the circuits. The slow speed is due to the technology used to construct the transistors in the circuit.

The technology used is called metal-gate CMOS, because the transistor gates are formed by metal deposition. More importantly, the gates are formed after the drain and source regions have been defined, and must overlap the source and drain somewhat to allow for alignment tolerances. This overlap plus the relatively large size of the transistors themselves result in high electrode capacitance, and that is what limits the speed of the circuit.

High speed CMOS became feasible with the development of the self-aligning silicon gate technology. In this process polysilicon gates are deposited **before** the source and drain regions are defined. Then the source and drain regions are formed by ion implantation using the gate itself as a mask for the implantation. This eliminates most of the overlap capacitance. In addition, the process allows smaller transistors. The result is a significant increase in circuit speed. The 74HC-series of CMOS logic circuits is based on this technology, and has speeds comparable to LS TTL, which is to say about 10 times faster than the 74C-series circuits.

The size reduction that contributes to the higher speed also demands an accompanying reduction in the maximum supply voltage. High-speed CMOS is generally limited to 6V.

WHAT IS CHMOS?

CHMOS is the name given to Intel's high-speed CMOS processes. There are two CHMOS processes, one based on an n-well structure and one based on a p-well structure. In the n-well structure, n-type wells are diffused into a p-type substrate. Then the n-channel transistors (nFETs) are built into the substrate and pFETs are built into the n-wells. In the p-well structure, p-type wells are diffused into an n-type substrate. Then the nFETs are built into the wells and pFETs, into the

substrate. Both processes have their advantages and disadvantages, which are largely transparent to the user.

Lower operating voltages are easier to obtain with the p-well structure than with the n-well structure. But the p-well structure does not easily adapt to an EPROM which would be pin-for-pin compatible with HMOS EPROMs. On the other hand the n-well structure can be based on the solidly founded HMOS process, in which nFETs are built into a p-type substrate. This allows somewhat more than half of the transistors in a CHMOS chip to be constructed by processes that are already well characterized.

Currently Intel's CHMOS microcontrollers and memory products are n-well devices, whereas CHMOS microprocessors are p-well devices.

Further discussion of the CHMOS technology is provided in References 1 and 2 (which are reprinted in the Microcontroller Handbook).

THE MCS®-51 FAMILY IN CHMOS

The 80C51BH is the CHMOS version of Intel's original 8051. The 80C31BH is the ROMless 80C51BH, equivalent to the 8031. These CHMOS devices are architecturally identical with their HMOS counterparts, except that they have two added features for reduced power. These are the Idle and Power Down modes of operation.

In most cases, an 80C51BH can directly replace the 8051 in existing applications. It can execute the same code at the same speed, accept signals from the same sources, and drive the same loads. However, the 80C51BH covers a wider range of speeds, will emit CMOS logic levels to CMOS loads, and will draw about 1/10 the current of an 8051 (and less yet in the reduced power modes). Interchangeability between the HMOS and CHMOS devices is discussed in more detail in the final section of this Application Note.

It should be noted that the 80C51BH CPU is not static. That means if the clock frequency is too low, the CPU might forget what it was doing. This is because the circuitry uses a number of dynamic nodes. A dynamic node is one that uses the node-to-ground capacitance to form a temporary storage cell. Dynamic nodes are used to reduce the transistor count, and hence the chip area, thus to produce a more economical device.

This is not to say that the on-chip RAM in CHMOS microcontrollers is dynamic. It's not. It's the CPU that is dynamic, and that is what imposes the minimum clock frequency specification.

LATCHUP

Latchup is an SCR-type turn-on phenomenon that is the traditional nemesis of CMOS systems. The substrate, the wells, and the transistors form parasitic pnpn structures within the device. These parasitic structures turn on like an SCR if a sufficient amount of forward current is driven through one of the junctions. From the circuit designer's point of view it can happen whenever an input or output pin is externally driven a diode drop above V_{CC} or below V_{SS} , by a source that is capable of supplying the required trigger current.

However much of a problem latchup has been in the past, it is good to know that in most recently developed CMOS devices, and specifically in CHMOS devices, the current required to trigger latchup is typically well over 100 mA. The 80C51BH is virtually immune to latchup. (References 1 and 2 present a discussion of the latchup mechanisms and the steps that are taken on the chip to guard against it.) Modern CMOS is not absolutely immune to latchup, but with trigger currents in the hundreds of mA, latchup is certainly a lot easier to avoid than it once was.

A careless power-up sequence might trigger a latchup in the older CMOS families, but it's unlikely to be a major problem in high-speed CMOS or in CHMOS. There is still some risk incurred in inserting or removing chips or boards in a CMOS system while the power is on. Also, severe transients, such as inductive kicks or momentary short-circuits, can exceed the trigger current for latchup.

For applications in which some latchup risk seems unavoidable, you can put a small resistor (100 Ω or so) in series with signal lines to ensure that the trigger current will never be reached. This also helps to control overshoot and RFI.

LOGIC LEVELS AND INTERFACING PROBLEMS

CMOS logic levels differ from TTL levels in two ways.

First, for equal supply voltages, CMOS gives (and requires) a higher "logic 1" level than TTL. Secondly, CMOS logic levels are V_{CC} (or V_{DD}) dependent, whereas guaranteed TTL logic levels are fixed when V_{CC} is within TTL specs.

Standard 74HC logic levels are as follows:

$$\begin{aligned} V_{IHMIN} &= 70\% \text{ of } V_{CC} \\ V_{ILMAX} &= 20\% \text{ of } V_{CC} \\ V_{OHMIN} &= V_{CC} - 0.1V, |I_{OH}| \leq 20 \mu A \\ V_{OLMAX} &= 0.1V, |I_{OL}| \leq 20 \mu A \end{aligned}$$

Figure 1 compares 74HC, LS TTL, and 74HCT logic levels with those of the HMOS 8051 and the CHMOS 80C51BH for $V_{CC} = 5V$.

Output logic levels depend of course on load current, and are normally specified at several load currents. When CMOS and TTL are powered by the same V_{CC} , the logic levels guaranteed on the data sheets indicate that CMOS can drive TTL, but TTL can't drive CMOS. The incompatibility is that the TTL circuit's V_{OH} level is too low to reliably be recognized by the CMOS circuit as a valid V_{IH} .

Since HMOS circuits were designed to be TTL-compatible, they have the same incompatibility.

Fortunately, 74HCT-series circuits are available to ease these interfacing problems. They have TTL-compatible logic levels at the inputs and standard CMOS levels at the outputs.

The 80C51BH is designed to work with either TTL or CMOS. Therefore its logic levels are specified very much like 74HCT circuits. That is, its input logic levels are TTL-compatible, and its output characteristics are like standard high-speed CMOS.

NOISE CONSIDERATIONS

One of the major reasons for going to CMOS has traditionally been that CMOS is less susceptible to noise. As previously noted, its low susceptibility to noise is

Logic State	$V_{CC} = 5V$				
	74HC	74HCT	LS TTL	8051	80C51BH
V_{IH}	3.5V	2.0V	2.0V	2.0V	1.9V
V_{IL}	1.0V	0.8V	0.8V	0.8V	0.9V
V_{OH}	4.9V	4.9V	2.7V	2.4V	4.5V
V_{OL}	0.1V	0.1V	0.5V	0.45V	0.45V

Figure 1. Logic Level Comparison. (Output voltage levels depend on load current. Data sheets list guaranteed output levels for several load currents. The output levels listed here are for minimum loading.)

partly due to superior noise margins, and partly due to its slow speed.

Noise margin is the difference between V_{OL} and V_{IL} , or between V_{OH} and V_{IH} . If V_{OH} from a driving circuit is 2.7V and V_{IH} to the driven circuit is 2.0V, then the driven circuit has 0.7V of noise margin at the logic high level. These kinds of comparisons show that an all-CMOS system has wider noise margins than an all-TTL system.

Figure 2 shows noise margins in CMOS and LS TTL systems when both have $V_{CC} = 5V$. It can be seen that CMOS/CMOS and CMOS/CHMOS systems have an edge over LS TTL in this respect.

Noise margins can be misleading, however, because they don't say how much noise energy it takes to induce in the circuit a noise voltage of sufficient amplitude to cause a logic error. This would involve consideration of the width of the noise pulse as compared with the circuit's response speed, and the impedance to ground from the point of noise introduction in the circuit.

When these considerations are included, it is seen that using the slower 74C- and 4000-series circuits with a 12 or 15V supply voltage does offer a truly improved level of noise immunity, but that high-speed CMOS at 5V is not significantly better than TTL.

One should not mistake the wider supply voltage tolerance of high-speed CMOS for V_{CC} glitch immunity. Supply voltage tolerance is a DC rating, not a glitch rating.

For any clocked CMOS, and most especially for VLSI CMOS, V_{CC} decoupling is critical. CHMOS draws

Interface	Noise Margin for $V_{CC} = 5V$	
	Logic Low $V_{IL} - V_{OL}$	Logic High $V_{OH} - V_{IH}$
74HC to 74HC	0.9V	1.4V
LSTTL to LSTTL	0.3V	0.7V
LSTTL to 74HCT	0.3V	0.7V
LSTTL to 80C51BH	0.3V	0.7V
74HC to 80C51BH	0.8V	3.0V
80C51BH to 74HC	0.8V	1.0V

Figure 2. Noise Margins for CMOS and LS TTL Circuits

current in extremely sharp spikes at the clock edges. The VHF and UHF components of these spikes are not drawn from the power supply, but from the decoupling capacitor. If the decoupling circuit is not sufficiently low in inductance, V_{CC} will glitch at each clock edge. We suggest that a 0.1 μF decoupler cap be used in a minimum-inductance configuration with the microcontroller. A minimum-inductance configuration is one that minimizes the area of the loop formed by the chip (V_{CC} to V_{SS}), the traces to the decoupler cap, and the decoupler cap. PCB designers too often fail to understand that if the traces that connect the decoupler cap to the V_{CC} and V_{SS} pins aren't short and direct, the decoupler loses much of its effectiveness.

Overshoot and ringing in signal lines are potential sources of logic upsets. These can largely be controlled by circuit layout. Inserting small resistors (about 100 Ω) in series with signal lines that seem to need them will also help.

The sharp edges produced by high-speed CMOS can cause RFI problems. The severity of these problems is largely a function of the PCB layout. We don't mean to imply that all RFI problems can be solved by a better PCB layout. It may well be, for example, that in some RFI-sensitive designs high-speed CMOS is simply not the answer. But circuit layout is a critical factor in the noise performance of any electronic system, and more so in high-speed CMOS systems than others.

Circuit layout techniques for minimizing noise susceptibility and generation are discussed in References 3 through 6.

UNUSED PINS

CMOS input pins should not be left to float, but should always be pulled to one logic level or the other. If they float, they tend to float into the transition region between 0 and 1, where the pullup and pulldown devices in the input buffer are both conductive. This causes a significant increase in I_{CC} . A similar effect exists in HMOS circuits, but with less noticeable results.

In 80C51BH and 80C31BH designs, unused pins of Ports 1, 2, and 3 can be ignored, because they have internal pullups that will hold them at a valid Logic 1 level. Port 0 pins are different, however, in not having internal pullups (except during bus operations).

When the 80C51BH is in reset, the Port 0 pins are in a float state unless they are externally pulled up or down. If it's going to be held in reset for just a short time, the transient float state can probably be ignored. When it comes out of reset, the pins stay afloat unless

they are externally pulled either up or down. Alternatively, the software can internally write 0s to whatever Port 0 pins may be unused.

The same considerations are applicable to the 80C31BH with regards to reset. But when the 80C31BH comes out of reset, it commences bus operations, during which the logic levels at the pins are always well defined as high or low.

Consider the 80C31BH in the Power Down or Idle modes, however. In those modes it is not fetching instructions, and the Port 0 pins will float if not externally pulled high or low. The choice of whether to pull them high or low is the designer's. Normally it is sufficient to pull them up to V_{CC} with 10k resistors. But if power is going to be removed from circuits that are connected to the bus, it will be advisable to pull the bus pins down (normally with 10k resistors). Considerations involved in selecting pullup and pulldown resistor values are as follows.

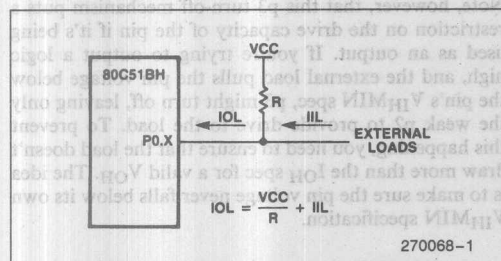


Figure 3a. Conditions defining the minimum value for R. P0.X is emitting a logic low. R must be large enough to not cause IOL to exceed data sheet specifications.

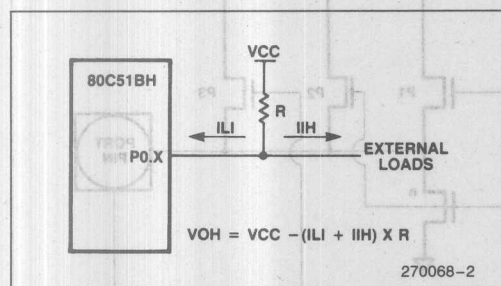


Figure 3b. Conditions defining the maximum value for R. P0.X is in a high impedance state. R must be small enough to keep VOH acceptably high.

PULLUP RESISTORS

If a pullup resistor is to be used on a Port 0 pin, its minimum value is determined by IOL requirements. If the pin is trying to emit a 0, then it will have to sink the current from the pullup resistor plus whatever other current may be sourced by other loads connected to the pin, as shown in Figure 3a, while maintaining a valid output low (VOL). To guarantee that the pin voltage will not exceed 0.45V, the resistor should be selected so that IOL doesn't exceed the value specified on the data sheet. In most CMOS applications, the minimum value would be about 2k Ω .

The maximum value you could use depends on how fast you want the pin to pull up after bus operations have ceased, and how high you want the VOH level to be. The smaller the resistor the faster it pulls up. Its effect on the VOH level is that $VOH = VCC - (ILI + IIL) \times R$. ILI is the input leakage current to the Port 0 pin, and IIL is the input high current to the external loads, as shown in Figure 3b. Normally VOH can be expected to reach 0.9 VCC if the pullup resistance does not exceed about 50k Ω .

Pulldown Resistors

If a pulldown resistor is to be used on a Port 0 pin, its minimum value is determined by VOH requirements during bus operations, and its maximum value is in most cases determined by leakage current.

During bus operations the port uses internal pullups to emit 1s. The D.C. Characteristics in the data sheet list guaranteed VOH levels for given IOH currents. (The "-" sign in the IOH value means the pin is sourcing that current to the external load, as shown in Figure 4.) To ensure the VOH level listed in the data sheet, the resis-

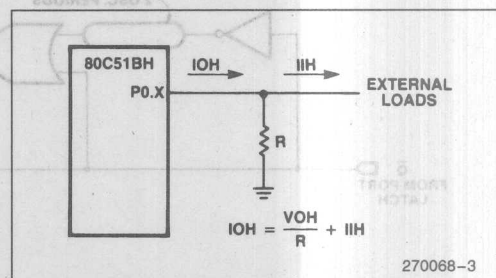


Figure 4a. Conditions defining the minimum value for R. P0.X is emitting a 1 in a bus operation. R must be large enough to not cause IOH to exceed data sheet specifications.

$$\frac{V_{OH}}{R} + I_{IH} \leq |I_{OH}|$$

tor has to satisfy where I_{IH} is the input high current to the external loads.

When the pin goes into a high impedance state, the pulldown resistor will have to sink leakage current from the pin, plus whatever other current may be sourced by other loads connected to the pin, as shown in Figure 4b. The Port 0 leakage current is I_{LI} on the data sheet. The resistor should be selected so that the voltage developed across it by these currents will be seen as a logic low by whatever circuits are connected to it (including the 80C51BH). In CMOS/CHMOS applications, 50k Ω is normally a reasonable maximum value.

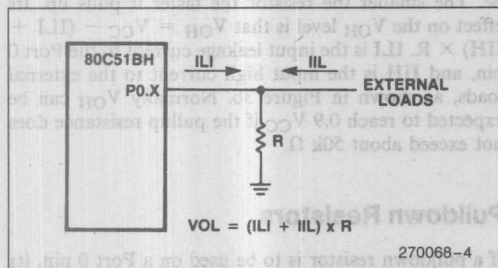


Figure 4b. Conditions defining the maximum value for R. P0.X is in a high impedance state.

R must be small enough to keep VOL acceptably low.

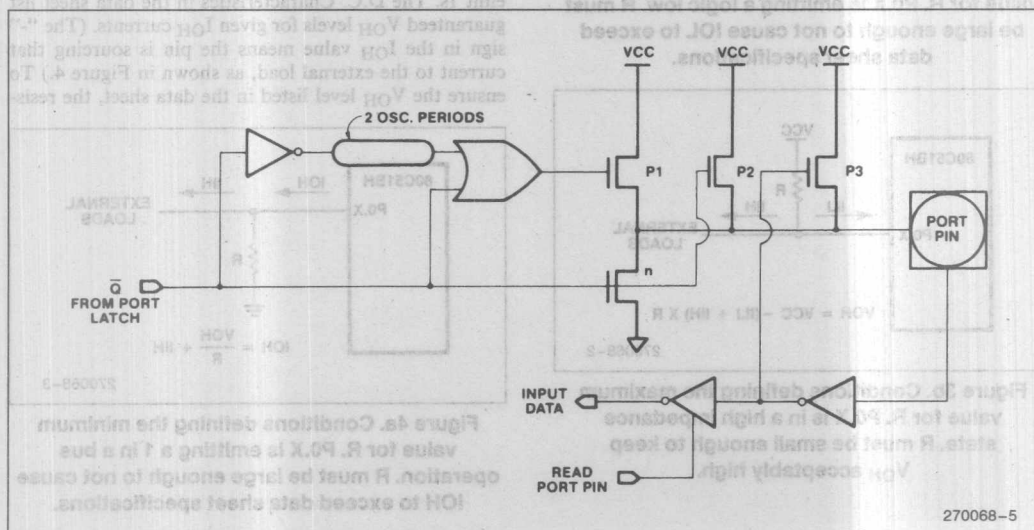


Figure 5. 80C51BH Output Drivers for Ports 1, 2 and 3

DRIVE CAPABILITY OF THE INTERNAL PULLUPS

There's an important difference between HMOS and CHMOS port drivers. The pins of Ports 1, 2, and 3 of the CHMOS parts each have three pullups: strong, normal, and weak, as shown in Figure 5. The strong pullup (p1) is only used during 0-to-1 transitions, to hasten the transition. The weak pullup (p2) is on whenever the bit latch contains a 1. The "normal" pullup (p3) is controlled by the pin voltage itself.

The reason that p3 is controlled by the pin voltage is that if the pin is being used as an input, and the external source pulls it to a low, then turning off p3 makes for a lower I_{IL} . The data sheet shows an " I_{TL} " specification. This is the current that p3 will source during the time the pin voltage is making its 1-to-0 transition. This is what I_{IL} would be if an input low at the pin didn't turn p3 off.

Note, however, that this p3 turn-off mechanism puts a restriction on the drive capacity of the pin if it's being used as an output. If you're trying to output a logic high, and the external load pulls the pin voltage below the pin's V_{IHMIN} spec, p3 might turn off, leaving only the weak p2 to provide drive to the load. To prevent this happening, you need to ensure that the load doesn't draw more than the I_{OH} spec for a valid V_{OH} . The idea is to make sure the pin voltage never falls below its own V_{IHMIN} specification.

POWER CONSUMPTION

The main reason for going to CMOS, of course, is to conserve power. (There are other reasons, but this is the main one.) Conserving power doesn't mean just reducing your electric bill. Nor does it necessarily relate to battery operation, although battery operation without CMOS is pretty unhandy. The main reason for conserving power is to be able to put more functionality into a smaller space. The reduced power consumption allows the use of smaller and lighter power supplies, and less heat being generated allows denser packaging of circuit components. Expensive fans and blowers can usually be eliminated.

A cooler running chip is also more reliable, since most random and wearout failures relate to die temperature. And finally, the lower power dissipation will allow more functions to be integrated onto the chip.

The reason CMOS consumes less power than NMOS is that when it's in a stable state there is no path of conduction from V_{CC} to V_{SS} except through various leakage paths. CMOS does draw current when it's changing states. How much current it draws depends on how often and how quickly it changes states.

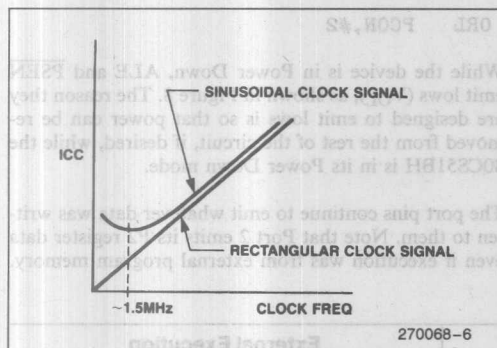


Figure 6. 80C51BH ICC vs. Clock Frequency

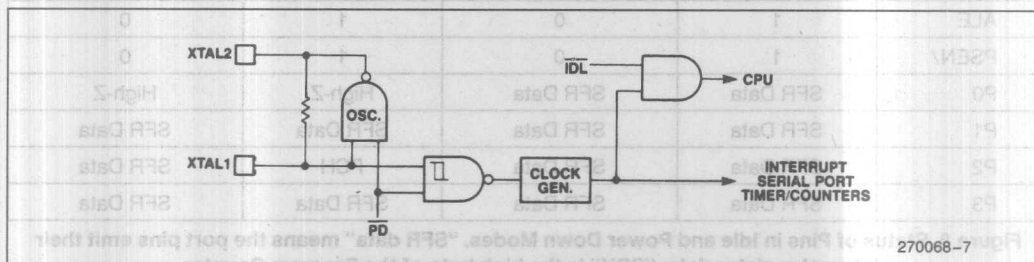


Figure 7. Oscillator and Clock Circuitry Showing Idle and Power Down Hardware

CMOS circuits draw current in sharp spikes during logical transitions. These current spikes are made up of two components. One is the current that flows during the transition time when pullup and pulldown FETs are both active. The average (DC) value of this component is larger when the transition times of the input signals are longer. For this reason, if the current draw is a critical factor in the design, slow rise and fall times should be avoided, even when the system speed doesn't seem to justify a need for nanosecond switching speeds.

The other component is the current that charges stray and load capacitance at the nodes of a CMOS logic gate. The average value of this current spike is its area (integral over time) multiplied by its rep rate. Its area is the amount of charge it takes to raise the node capacitance, C , to V_{CC} . That amount of charge is just $C \times V_{CC}$. So the average value of the current spike is $C \times V_{CC} \times f$, where f is the clock frequency.

This component of current increases linearly with clock frequency. For minimal current draw, the 80C52BH-2 is spec'd to run at frequencies as low as 500 kHz.

Keep in mind, though, that other component of current that is due to slow rise and fall times. A sinusoid is not the optimal waveform to drive the XTAL1 pin with. Yet crystal oscillators, including the one on the 80C51BH, generate sinusoidal waveforms. Therefore, if the on-chip oscillator is being used, you can expect the device to draw more current at 500 kHz, than it does at 1.5 MHz, as shown in Figure 6. If you derive a good sharp square wave from an external oscillator, and use that to drive XTAL1, then the microcontroller will draw less current. But the external oscillator will probably make up the difference.

The 80C51BH has two power-saving features not available in the HMOS devices. These are the Idle and Power Down modes of operation. The on-chip hardware that implements these reduced power modes is shown in Figure 7. Both modes are invoked by software.

Idle: In the Idle Mode ($\overline{IDL} = 0$ in Figure 7), the CPU puts itself to sleep by gating off its own clock. It doesn't stop the oscillator. It just stops the internal clock signal from getting to the CPU. Since the CPU draws 80 to 90 percent of the chip's power, shutting it off represents a fairly significant power savings. The on-chip peripherals (timers, serial port, interrupts, etc.) and RAM continue to function as normal. The CPU status is preserved in its entirety: the Stack Pointer, Program Counter, Program Status Word, Accumulator, and all other registers maintain their data during Idle.

The Idle Mode is invoked by setting bit 0 (IDL) of the PCON register. PCON is not bit-addressable, so the bit has to be set by a byte operation, such as

```
ORL PCON,#1
```

The PCON register also contains flag bits GF0 and GF1, which can be used for any general purposes, or to give an indication if an interrupt occurred during normal operation or during Idle. In this application, the instruction that invokes Idle also sets one or both of the flag bits. Their status can then be checked in the interrupt routines.

While the device is in the Idle Mode, ALE and \overline{PSEN} emit logic high (V_{OH}), as shown in Figure 8. This is so external EPROM can be deselected and have its output disabled.

The port pins hold the logical states they had at the time the Idle was activated. If the device was executing out of external program memory, Port 0 is left in a high impedance state and Port 2 continues to emit the high byte of the program counter (using the strong pullups to emit 1s). If the device was executing out of internal program memory, Ports 0 and 2 continue to emit whatever is in the P0 and P2 registers.

There are two ways to terminate Idle. Activation of any enabled interrupt will cause the hardware to clear bit 0 of the PCON register, terminating the Idle mode. The interrupt will be serviced, and following RETI the next instruction to be executed will be the one following the instruction that invoked Idle.

The other way is with a hardware reset. Since the clock oscillator is still running, RST only needs to be held active for two machine cycles (24 oscillator periods) to complete the reset. Note that this exit from Idle writes 1s to all the ports, initializes all SFRs to their reset values, and restarts program execution from location 0.

Power Down: In the Power Down Mode ($\overline{PD} = 0$ in Figure 7), the CPU puts the whole chip to sleep by turning off the oscillator. In case it was running from an external oscillator, it also gates off the path to the internal phase generators, so no internal clock is generated even if the external oscillator is still running. The on-chip RAM, however, saves its data, as long as V_{CC} is maintained. In this mode the only I_{CC} that flows is leakage, which is normally in the micro-amp range.

The Power Down Mode is invoked by setting bit 1 in the PCON register, using a byte instruction such as

```
ORL PCON,#2
```

While the device is in Power Down, ALE and \overline{PSEN} emit lows (V_{OL}), as shown in Figure 8. The reason they are designed to emit lows is so that power can be removed from the rest of the circuit, if desired, while the 80CS51BH is in its Power Down mode.

The port pins continue to emit whatever data was written to them. Note that Port 2 emits its P2 register data even if execution was from external program memory.

Pin	Internal Execution		External Execution	
	Idle	Power Down	Idle	Power Down
ALE	1	0	1	0
$\overline{PSEN}/$	1	0	1	0
P0	SFR Data	SFR Data	High-Z	High-Z
P1	SFR Data	SFR Data	SFR Data	SFR Data
P2	SFR Data	SFR Data	PCH	SFR Data
P3	SFR Data	SFR Data	SFR Data	SFR Data

Figure 8. Status of Pins in Idle and Power Down Modes. "SFR data" means the port pins emit their internal register data. "PCH" is the high byte of the Program Counter.

Port 0 also emits its P0 register data, but if execution was from external program memory, the P0 register data is FF. The oscillator is stopped, and the part remains in this state as long as V_{CC} is held, and until it receives an external reset signal.

The only exit from Power Down is a hardware reset. Since the oscillator was stopped, RST must be held active long enough for the oscillator to re-start and stabilize. Then the reset function initializes all the Special Function Registers (ports, timers, etc.) to their reset values, and re-starts the program from location 0. Therefore, timer reloads, interrupt enables, baud rates, port status, etc. need to be re-established. Reset does not affect the content of the on-chip data RAM. If V_{CC} was held during Power Down, the RAM data is still good.

USING THE POWER DOWN MODE

The software-invoked Power Down feature offers a means of reducing the power consumption to a mere trickle in systems which are to remain dormant for some period of time, while retaining important data.

The user should give some thought to what state the port pins should be left in during the time the clock is stopped, and write those values to the port latches before invoking Power Down.

If V_{CC} is going to be held to the entire circuit, one would want to write values to the port latches that would deselect peripherals before invoking Power Down. For example, if external memory is being used, the P2 SFR should be loaded with a value which will not generate an active chip select to any memory device.

In some applications, V_{CC} to part of the system may be shut off during Power Down, so that even quiescent and standby currents are eliminated. Signal lines that connect to those chips must be brought to a logic low, whether the chip in question is CMOS, NMOS, or TTL, before V_{CC} is shut off to them. CMOS pins have parasitic pn junctions to V_{CC} , which will be forward biased if V_{CC} is reduced to zero while the pin is held at a logic high. NMOS pins often have FETs that look like diodes to V_{CC} . TTL circuits may actually be damaged by an input high if $V_{CC} = 0$. That's why the 80C51BH outputs lows at ALE and \overline{PSEN} during Power Down.

Figure 9 shows a circuit that can be used to turn V_{CC} off to part of the system during Power Down. The circuit will ensure that the secondary circuit is not de-energized until after the 80C31BH is in Power Down, and that the 80C31BH does not receive a reset (terminating the Power Down mode) before the secondary circuit is re-energized. Therefore, the program memory itself can be part of the secondary circuit.

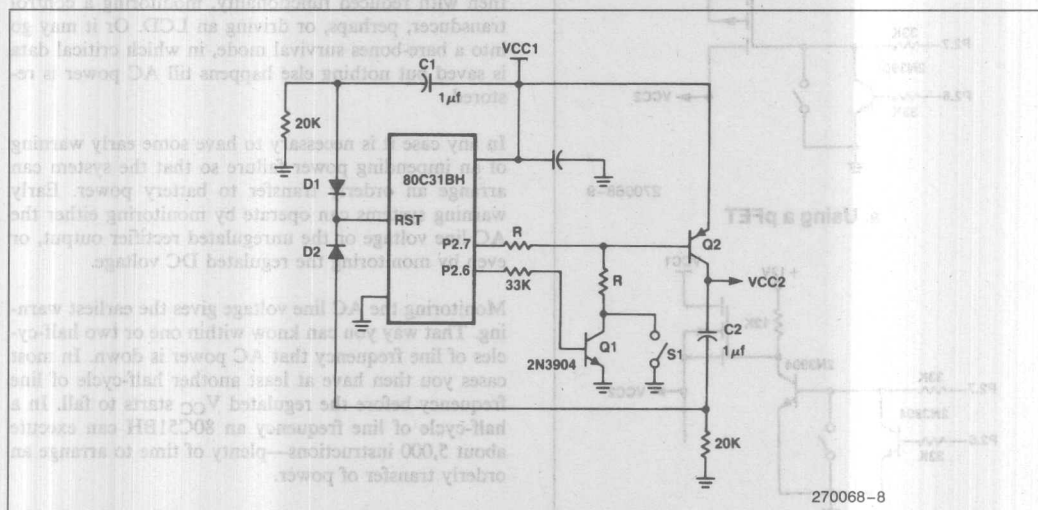


Figure 9. The 80C31BH de-energizes part of the circuit (VCC2) when it goes into Power Down. Selections of R and Q2 depend on VCC2 current draw.

In Figure 9, when V_{CC} is switched on to the 80C31BH, capacitor C1 provides a power-on reset. The reset function writes 1s to all the port pins. The 1 at P2.6 turns Q1 on, enabling V_{CC} to the secondary circuit through transistor Q2. As the 80C31BH comes out of reset, Port 2 commences emitting the high byte of the Program Counter, which results in the P2.7 and P2.6 pins outputting 0s. The 0 at P2.7 ensures continuation of V_{CC} to the secondary circuit.

The system software must now write a 1 to P2.7 and a 0 to P2.6 in the Port 2 SFR, P2. These values will not appear at the Port 2 pins as long as the device is fetching instructions from external program memory. However, whenever the 80C31BH goes into Power Down, these values will appear at the port pins, and will shut off both transistors, disabling V_{CC} to the secondary circuit.

Closing the switch S1 re-energizes the secondary circuit, and at the same time sends a reset through C2 to the 80C31BH to wake it up. The diode D1 is to prevent C1 from hogging current from C2 during this secondary reset. D2 prevents C2 from discharging through the RST pin when V_{CC} to the secondary circuit goes to zero.

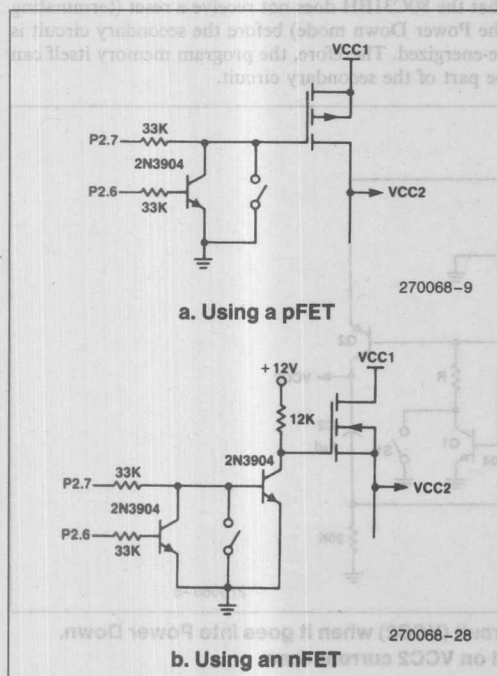


Figure 10. Using Power MOSFETs to Control V_{CC2}

USING POWER MOSFETs to CONTROL V_{CC}

Power MOSFETs are gaining in popularity (and availability). The easiest way to control V_{CC} is with a Logic Level pFET, as shown in Figure 10a. This circuit allows the full V_{CC} to be used to turn the device on. Unfortunately, power pFETs are not economically competitive with bipolar transistors of comparable ratings.

Power nFETs are both economical and available, and can be used in this application if a DC supply of higher voltage is available to drive the gate. Figure 10b shows how to implement a V_{CC} switch using a power nFET and a (nominally) +12V supply. The problem here is that if the device is on, its source voltage is +5V. To maintain the on state, the gate has to be another 5 or 10V above that. The "12V" supply is not particularly critical. A minimally filtered, unregulated rectifier will suffice.

BATTERY BACKUP SYSTEMS

Here we consider circuits that normally draw power from the AC line, but switch to battery operation in the event of a power failure. We assume that in battery operation high-current loads will be allowed to die along with the AC power. The system may continue then with reduced functionality, monitoring a control transducer, perhaps, or driving an LCD. Or it may go into a bare-bones survival mode, in which critical data is saved but nothing else happens till AC power is restored.

In any case it is necessary to have some early warning of an impending power failure so that the system can arrange an orderly transfer to battery power. Early warning systems can operate by monitoring either the AC line voltage or the unregulated rectifier output, or even by monitoring the regulated DC voltage.

Monitoring the AC line voltage gives the earliest warning. That way you can know within one or two half-cycles of line frequency that AC power is down. In most cases you then have at least another half-cycle of line frequency before the regulated V_{CC} starts to fall. In a half-cycle of line frequency an 80C51BH can execute about 5,000 instructions—plenty of time to arrange an orderly transfer of power.

The circuit in Figure 11 uses a Zener diode to test the line voltage each half-cycle, and a junction transistor to pass the information on to the 80C51BH. (Obviously a voltage comparator with a suitable reference source can

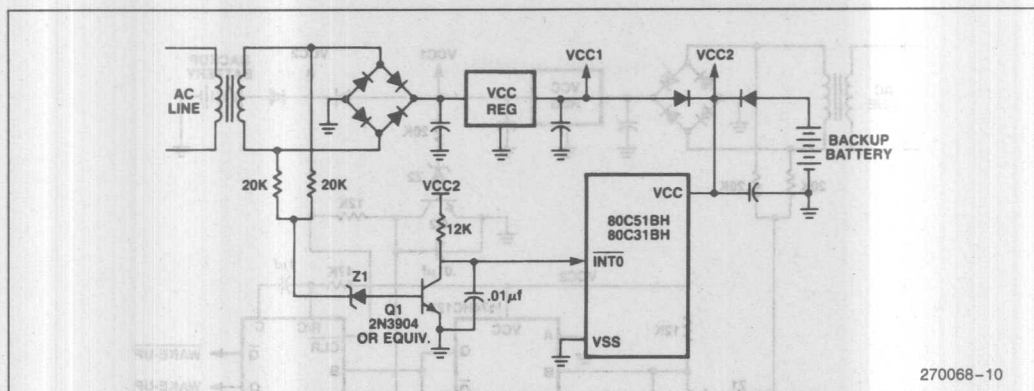


Figure 11. Power Failure Detector with Battery Backup. When AC power fails, VCC1 goes down and VCC2 is held.

perform the same function, if one prefers.) The way it works is if the line voltage reaches an acceptably high level, it breaks over Z1, drives Q1 to saturation, and interrupts the 80C51BH. The interrupt would be transition-activated, in this application. The interrupt service routine reloads one of the C51BH's timers to a value that will make it roll over in something between one and two half-cycles of line frequency. As long as the line voltage is healthy, the timer never rolls over, because it is reloaded every half-cycle. If there is a single half-cycle in which the line voltage doesn't reach a high enough level to generate the interrupt, the timer rolls over and generates a timer interrupt.

The timer interrupt then commences the transition to battery backup. Critical data needs to be copied into protected RAM. Signals to circuits that are going to lose power must be written to logic low. Protected circuits (those powered by VCC2) that communicate with unprotected circuits must be deselected. The microcontroller itself may be put into Idle, so that it can continue some level of interrupt-driven functionality, or it may be put into Power Down.

Note that if the CPU is going to invoke Power Down, the Special Function Registers may also need to be copied into protected RAM, since the reset that terminates the Power Down mode will also initialize all the SFRs to their reset values.

The circuit in Figure 11 does not show a wake-up mechanism. A number of choices are available, however. A pushbutton could be used to generate an interrupt, if the CPU is in Idle, or to activate reset, if the CPU is in Power Down.

Automatic wake-up on power restoration is also possible. If the CPU is in Idle, it can continue to respond to any interrupts that might be generated by Q1. The interrupt service routine determines from the status of flag bits GF0 and GF1 in PCON that it is in Idle because there was a power outage. It can then sample VCC1 through a voltage comparator similar to Z1, Q1 in Figure 11. A satisfactory level of VCC1 would be indicated by the transistor being in saturation.

But perhaps you can't spare the timer that is the key to the operation of the circuit in Figure 11. In that case a retriggerable one-shot, triggered by the AC line voltage, can perform essentially the same function. Figure 12 shows an example of this type of power failure detector. A retriggerable one-shot (one half of a 74HC123) monitors the AC line voltage through transistor Q1. Q1 re-triggers the one-shot every half-cycle of line frequency. If the output pulse width is between one and two half-cycles of line frequency, then a single missing or low half-cycle will generate an active low warning flag, which can be used to interrupt the microcontroller.

The interrupt routine takes care of the transition to battery backup. From this point VCC1 may or may not actually drop out. The missing half-cycle of line voltage that caused the power down sequence may have been nothing more than a short glitch. If the AC line comes back strong enough to trigger the one-shot while VCC1 is still up (as indicated by the state of transistor Q2), then the other half of the 74HC123 will generate a wake-up signal.

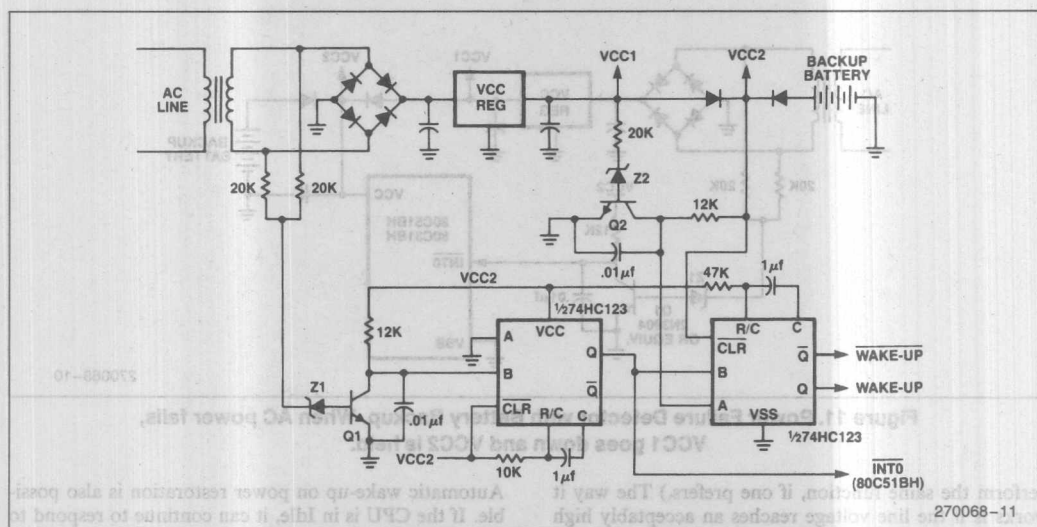


Figure 12. Power Failure Detector uses retriggerable one-shots to flag impending power outage and generate automatic wake-up when power returns.

Having been awakened, the 80C51BH will stay awake for at least another half-cycle of line frequency (another 5,000 or so instructions) before possibly being told to arrange another transfer of power. Consequently, if the line voltage is jittering erratically around the switchover point (determined by diode Z1), the system will limp along executing in half-cycle units of line frequency.

On the other hand, if the power outage is real and lengthy, VCC1 will eventually fall below the level at which the backup battery takes over. The backup battery maintains power to the 80C51BH, and to the 74HC123, and to whatever other circuits are being protected during this outage. The battery voltage must be high enough to maintain VCCMIN specs to the 80C51BH.

If the microcontroller is an 80C31BH, executing out of external ROM, and if the C31BH is put into Idle during the power outage, then the external ROM must also be supplied by the battery. On the other hand, if the C31BH is put into Power Down during the outage, then the ROM can be allowed to die with the AC power. The considerations here are the same as in Figure 9: VCC to the ROM is still up at the time Power Down is invoked, and we must ensure (through selection of diode Z2 in Figure 12) that the 80C31BH is not awakened till ROM power is back in spec.

POWER SWITCHOVER CIRCUITS

Battery backup systems need to have a way for the protected circuits to draw power from the line-operated power supply when that source is available, and to switch over to battery power when required. The switchover circuit is simple if the entire system is to be battery powered in the event of a line power outage. In that case a pair of diodes suffice, as shown in Figure 12, provided VCCMIN specs are still met after the diode drop has been subtracted from its respective power source.

The situation becomes more complicated when part of the circuit is going to be allowed to die when the AC power goes out. In that case it is difficult to maintain equal VCCs to protected and unprotected circuits (and possibly dangerous not to).

The problem can be alleviated by using a Schottky diode instead of a 1N4001, for its lower forward voltage drop. The 1N5820, for example, has a forward drop of about 0.35V at 1A.

Other solutions are to use a transistor or power MOSFET switch, as shown in Figure 13. With minor modifications this switch can be controlled by port pins.

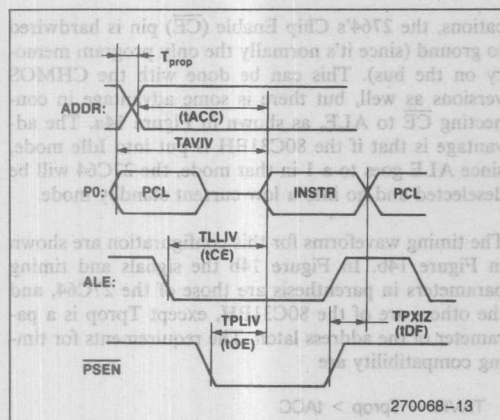


Figure 14b. Timing Waveforms for 80C31BH + 27C64

select the EPROM when the CPU is in Power Down. If Idle is never invoked, \overline{CE} of the EPROM can be connected to P2.7 of the 80C31BH, as shown in Figure 15a. In normal operation, P2.7 will be emitting the MSB of the Program Counter, which is 0 if the program contains less than 32K of code. Then when the CPU goes into Power Down, the Port 2 pins emit P2 SFR data, which puts a 1 at P2.7, thus deselecting the EPROM.

If Idle and Power Down are both going to be used, \overline{CE} of the EPROM can be driven by the logical OR of ALE and P2.7, as shown in Figure 15b. In Idle, $ALE = 1$ will deselect the EPROM, and in Power Down, P2.7 = 1 will deselect it.

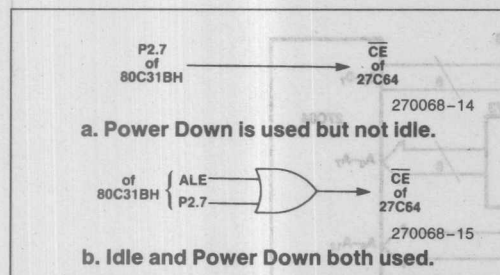


Figure 15. Modifications to 80C31BH/27C64 Interface

Pulldown resistors are shown in Figure 14a under the assumption that something on the bus is going to have its V_{CC} removed during Power Down. If this is not the case, pullups can be used as well as pulldowns.

The 87C64 is like the 27C64 except that it has an on-chip address latch. The Port 0 pins are tied to both address and data pins of the 87C64, as shown in Figure 16a. ALE drives the EPROM's ALE/ \overline{CS} input. During ALE high, the address information is allowed to flow into the EPROM and begin accessing the code byte. On the falling edge of ALE the address byte is internally latched. The A0-A7 inputs are then ignored and the same bus lines are used to transmit the fetched code byte from the O0-O7 pins back to the 80C31BH.

The timing waveforms for this configuration are shown in Figure 16b. In Figure 16b the signals and timing parameters in parentheses are those of the 87C64, and the others are of the 80C31BH. The requirements for timing compatibility are

$$TLHLL > tLL$$

$$TAVLL > tAL$$

$$TLLAX > tLA$$

$$TLLIV > tACL$$

$$TPLIV > tOE$$

$$TLLPL > tCOE$$

$$TPXIZ > tOHZ$$

The same considerations apply to the 87C64 as to the 27C64 with regards to the Idle and Power Down modes. Basically you want $\overline{CS} = 1$ if V_{CC} is maintained to the EPROM, and $\overline{CS} = \overline{OE} = 0$ if V_{CC} is removed.

SCANNING A KEYBOARD

There are many different kinds of keyboards, but alphanumeric keyboards generally consist of a matrix of 8 scan lines and 8 receive lines as shown in Figure 17. Each set of lines connects to one port of the microcontroller. The software has written 0s to the scan lines, and 1s to the receive lines. Pressing a key connects a scan line to a receive line, thus pulling the receive line to a logic low.

The 8 receive lines are ANDed to one of the external interrupt pins, so that pulling any of the receive lines low generates an interrupt. The interrupt service routine has to identify the pressed key, if only one key is down, and convert that information to some useful output. If more than one key in the line matrix is found to be pressed, no action is taken. (This is a "two key lock-out" scheme.)

On some keyboards, certain keys (Shift, Control, Escape, etc.) are not a part of the line matrix. These keys would connect directly to a port pin on the microcontroller, and would not cause lock-out if pressed simultaneously with a matrix key, nor generate an interrupt if pressed singly.

Normally the microcontroller would be in idle mode when a key has not been pressed, and another task is not in progress. Pressing a matrix key generates an inter-

rupt, which terminates the Idle. The interrupt service routine would first call a 30 ms (or so) delay to debounce the key, and then set about the task of identifying which key is down.

First, the current state of the receive lines is latched into an internal register. If a single key is down, all but one of the lines would be read as 1s. Then 0s are written to the receive lines and 1s to the scan lines, and the scan lines are read. If a single key is down, all but one of

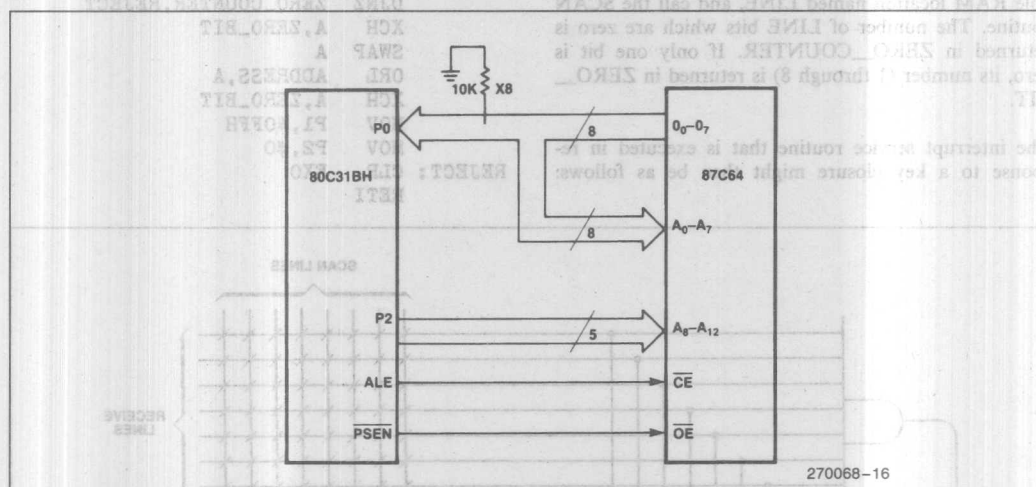


Figure 16a. 80C31BH + 87C64

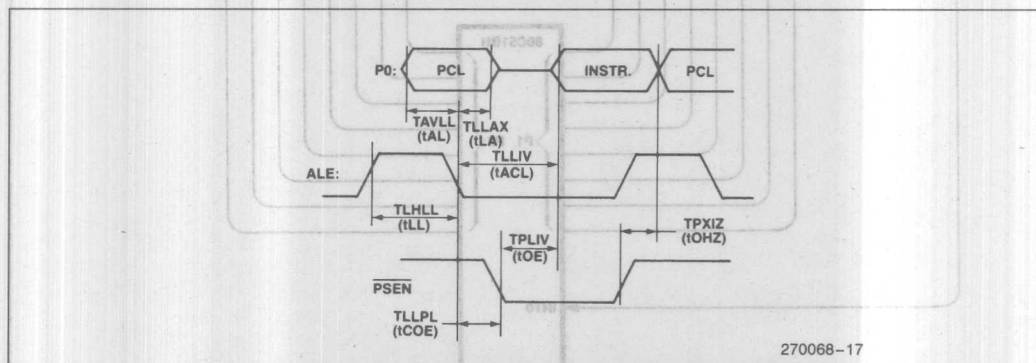


Figure 16b. Timing Waveforms for 80C31BH + 87C64

these lines would be read as 1s. By locating the single 0 in each set of lines, the pressed key can be identified. If more than one matrix key is down, one or both sets of lines will contain multiple 0s.

A subroutine is used to determine which of 8 bits in either set of lines is 0, and whether more than one bit is 0. Figure 18 shows a subroutine (SCAN) which does that using the 8051's bit-addressing capability. To use the subroutine, move the line data into a bit-addressable RAM location named LINE, and call the SCAN routine. The number of LINE bits which are zero is returned in ZERO_COUNTER. If only one bit is zero, its number (1 through 8) is returned in ZERO_BIT.

The interrupt service routine that is executed in response to a key closure might then be as follows:

```

RESPONSE_TO_KEY_CLOSURE:
CALL DEBOUNCE_DELAY
MOV LINE,P1; ;See Figure 17.
CALL SCAN
DJNZ ZERO_COUNTER,REJECT
MOV ADDRESS,ZERO_BIT
MOV P2,#0FFH; ;See Figure 17.
MOV P1,#0
MOV LINE,P2
CALL SCAN
DJNZ ZERO_COUNTER,REJECT
XCH A,ZERO_BIT
SWAP A
ORL ADDRESS,A
XCH A,ZERO_BIT
MOV P1,#0FFH
MOV P2,#0
REJECT: CLR EX0
RETI

```

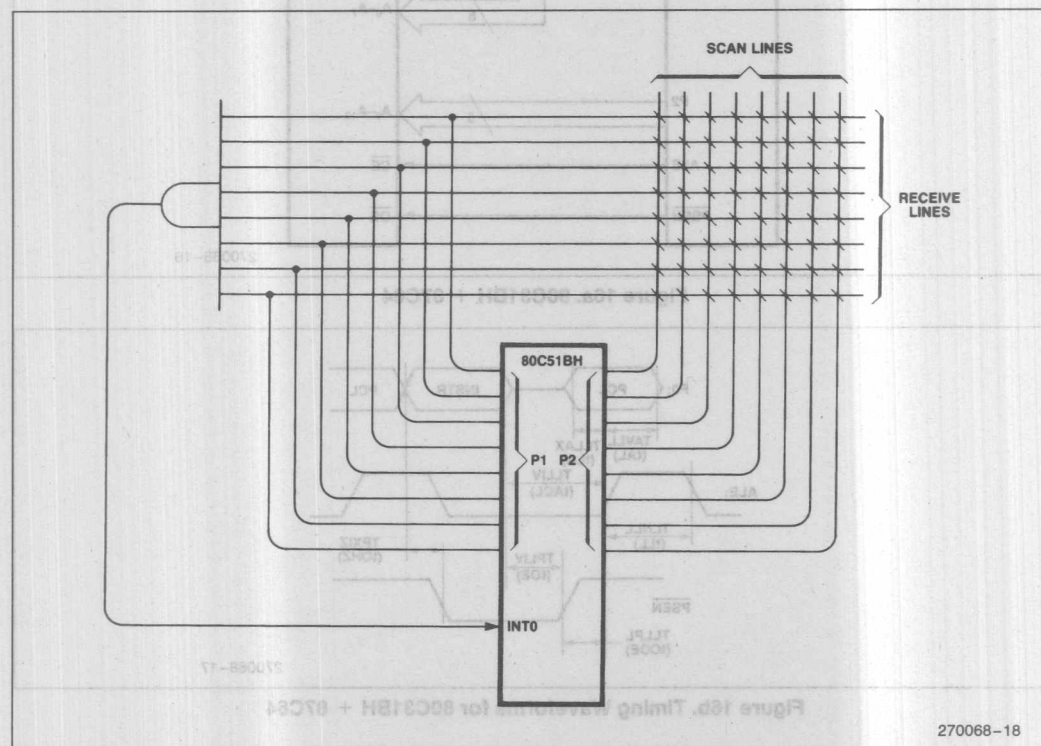


Figure 17. Scanning a Keyboard

SCAN:	MOV	ZERO_COUNTER,#0	; ZERO_COUNTER counts the number of 0s in LINE.
	JB	LINE_0.ONE	; Test LINE bit 0.
	INC	ZERO_COUNTER	; If LINE_0 = 0, increment ZERO_COUNTER
	MOV	ZERO_BIT,#1	; and record that line number 1 is active.
ONE:	JB	LINE_1.TWO	; Procedure continues for other LINE bits.
	INC	ZERO_COUNTER	
	MOV	ZERO_BIT,#2	; Line number 2 is active.
TWO:	JB	LINE_2.THREE	
	INC	ZERO_COUNTER	
	MOV	ZERO_BIT,#3	; Line number 3 is active.
THREE:	JB	LINE_3.FOUR	
	INC	ZERO_COUNTER	
	MOV	ZERO_BIT,#4	; Line number 4 is active.
FOUR:	JB	LINE_4.FIVE	
	INC	ZERO_COUNTER	
	MOV	ZERO_BIT,#5	; Line number 5 is active.
FIVE:	JB	LINE_5.SIX	
	INC	ZERO_COUNTER	
	MOV	ZERO_BIT,#6	; Line number 6 is active.
SIX:	JB	LINE_6.SEVEN	
	INC	ZERO_COUNTER	
	MOV	ZERO_BIT,#7	; Line number 7 is active.
SEVEN:	JB	LINE_7.EIGHT	
	INC	ZERO_COUNTER	
	MOV	ZERO_BIT,#8	; Line number 8 is active.
EIGHT:	RET		

270068-19

Figure 18. Subroutine SCAN Determines Which of 8 Bits in LINE is Zero

Notice that `RESPONSE_TO_KEY_CLOSURE` does not change the Accumulator, the PSW, nor any of the registers R0 through R7. Neither do `SCAN` or `DEBOUNCE_DELAY`.

What we come out with then is a one-byte key address (`ADDRESS`) which identifies the pressed key. The key's scan line number is in the upper nibble of `ADDRESS`, and its receive line number is in the lower nibble. `ADDRESS` can be used in a look-up table to generate a key code to transmit to a host computer, and/or to a display device.

The keyboard interrupt itself must be edge-triggered, rather than level-activated, so that the interrupt routine is invoked when a key is pressed, and is not constantly being repeated as long as the key is held down. In edge-triggered mode, the on-chip hardware clears the interrupt flag (`EX0`, in this case) as the service routine is being vectored to. In this application, however, contact bounce will cause several more edges to occur after the service routine has been vectored to, during the `DEBOUNCE_DELAY` routine. Consequently it is necessary to clear `EX0` again in software before executing `RETI`.

The debounce delay routine also takes advantage of the Idle mode. In this routine a timer must be preloaded with a value appropriate to the desired length of delay. This would be

$$\text{timer preload} = \frac{(\text{osc kHz}) \times (\text{delay time ms})}{12}$$

For example, with a 3.58 MHz oscillator frequency, a 30 ms delay could be obtained using a preload value of -8950, or DD0A, in hex digits.

In the debounce delay routine (Figure 19), the timer interrupt is enabled and set to a higher priority than the keyboard interrupt, because as we invoke Idle, the keyboard interrupt is still "in progress". An interrupt of the same priority will not be acknowledged, and will not terminate the Idle mode. With the timer interrupt set to priority 1, while the keyboard interrupt is a priority 0, the timer interrupt, when it occurs, will be acknowledged and will wake up the CPU. The timer interrupt service routine does not itself have to do anything. The service routine might be nothing more than a single `RETI` instruction. `RETI` from the timer interrupt service routine then returns execution to the debounce delay routine, which shuts down the timer and returns execution to the keyboard service routine.

DRIVING AN LCD

An LCD (Liquid Crystal Display) consists of a backplane and any number of segments or dots which will be used to form the image being displayed. Applying a voltage (nominally 4 or 5V) between any segment and the backplane causes the segment to darken. The only catch is that the polarity of the applied voltage has to be periodically reversed, or else a chemical reac-

```

DEBOUNCE_DELAY:
MOV     TL1,#TL1_PRELOAD ; Preload low byte.
MOV     TH1,#TH1_PRELOAD ; Preload high byte.
SETB    ET1               ; Enable Timer 1 interrupt.
SETB    PT1               ; Set Timer 1 interrupt to high priority.
SETB    TR1               ; Start timer running.
ORL     PCON,#1           ; Invoke Idle mode.

; The next instruction will not be executed until the delay times out.

CLR     TR1               ; Stop the timer.
CLR     PT1               ; Back to priority 0 (if desired).
CLR     ET1               ; Disable Timer 1 interrupt (if desired).
RET                                ; Continue keyboard scan

```

270068-20

Figure 19. Subroutine DEBOUNCE__DELAY Puts the 80C51BH into Idle During the Delay Time

tion takes place in the LCD which causes deterioration and eventual failure of the liquid crystal.

To prevent this happening, the backplane and all the segments are driven with an AC signal, which is derived from a rectangular voltage waveform. If a segment is to be "off" it is driven by the same waveform as the backplane. Thus it is always at backplane potential. If the segment is to be "on" it is driven with a waveform that is the inverse of the backplane waveform. Thus it has about 5V of periodically changing polarity between it and the backplane.

With a little software overhead, the 80C51BH can perform this task without the need for additional LCD drivers. The only drawback is that each LCD segment uses up one port pin, and the backplane uses one more. If more than, say, two 7-segment digits are being driven, there aren't many port pins left for other tasks. Nevertheless, assuming a given application leaves enough port pins available to support this task, the considerations for driving the LCD are as follows.

Suppose, for example, it is a 2-digit display with a decimal point. One port (TENS__DIGIT) connects to the 7 segments of the tens digit plus the backplane. Another port (ONES__DIGIT) connects to a decimal point plus the 7 segments of the ones digit.

One of the 80C51BH's timers is used to mark off half-periods of the drive voltage waveform. The LCD drive waveform should have a rep rate between 30 and 100 Hz, but it's not very critical. A half-period of 12 ms will set the rep rate to about 42 Hz. The preload/reload value to get 12 ms to rollover is the 2's complement negative of the oscillator frequency in kHz: if the oscillator frequency is 3.58 MHz, the reload value is -3580, or F204 in hex digits.

Now, the 80C51BH would normally be in Idle, to conserve power, during the time that the LCD and other

tasks are not requiring servicing. When the timer rolls over it generates an interrupt, which brings the 80C51BH out of Idle. The service routine reloads the timer (for the next rollover), and inverts the logic levels of all the pins that are connected to the LCD. It might look like this:

```

LCD_DRIVE_INTERRUPT:
MOV     TL1,#LOW( - XTAL_FREQ)
MOV     TH1,#HIGH( - XTAL_FREQ)
XRL     TENS_DIGIT,#0FFH
XRL     ONES_DIGIT,#0FFH
RET

```

To update the display, one would use a look-up table to generate the characters. In the table, "on" segments are represented as 1s, and "off" segments as 0s. The backplane bit is represented as a 0. The quantity to be displayed is stored in RAM as a BCD value. The look-up table operates on the low nibble of the BCD value, and produces the bit pattern that is to be written to either the ones digit or the tens digit. Before the new patterns can be written to the LCD, the LCD drive interrupt has to be disabled. That is to prevent a polarity reversal from taking place between the times the two digits are written. An update subroutine is shown in Figure 20.

USING AN LCD DRIVER

As was noted, driving an LCD directly with an 80C51BH uses a lot of port pins. LCD drivers are available in CMOS to interface an 80C51BH to a 4-digit display using only 7 of the C51BH's I/O pins. Basically, the C51BH tells the LCD driver what digit is to be displayed (4 bits) and what position it is to be displayed in (2 bits), and toggles a Chip Select pin to tell the driver to latch this information. The LCD driver generates the display characters (hex digits), and takes care of the polarity reversals using its own RC oscillator to generate the timing.

Figure 25 shows an 80C51BH working with an ICM7211M to drive a 4-digit LCD, and the software that updates the display.

One could equally well send information to the LCD driver over the bus. In that case, one would set up the Accumulator with the digit select and data input bits, and execute a MOVX@ R0,A instruction. The LCD driver's chip select would be driven by the CPU's WR signal. This is a little easier in software than the direct bit manipulation shown in Figure 21. However, it uses more I/O pins, unless there is already some external memory involved. In that case, no extra pins are used up by adding the LCD driver to the bus.

RESONANT TRANSDUCERS

Analog transducers are often used to convert the value of a physical property, such as temperature, pressure, etc., to an analog voltage. These kinds of transducers then require an analog-to-digital converter to put the measurement into a form that is compatible with a digital control system. Another kind of transducer is now becoming available that encodes the value of the physical property into a signal that can be directly read by a digital control system. These devices are called resonant transducers.

Resonant transducers are oscillators whose frequency depends in a known way on the physical property being measured. These devices output a train of rectangular pulses whose repetition rate encodes the value of the quantity being measured. The pulses can in most cases be fed directly into the 80C51BH, which then measures either the frequency or period of the incoming signal, basing the measurement on the accuracy of its own clock oscillator. The 80C51BH can even do this in its sleep; that is, in Idle.

When the frequency or period measurement is completed, the C51BH wakes itself up for a very short time to perform a sanity check on the measurement and convert it in software to any scaling of the measured quantity that may be desired. The software conversion can include corrections for nonlinearities in the transducer's transfer function.

Resolution is also controlled by software, and can even be dynamically varied to meet changing needs as a situation becomes more critical. For example, in a process controller you can increase your resolution ("fine tune" the control, as it were) as the process approaches its target.

The nominal reference frequency of the output signal from these devices is in the range of 20 Hz to 500 kHz, depending on the design. Transducers are available that have a full scale frequency shift 2 to 1. The transducer operates from a supply voltage range of 3V to 20V, which means it can operate from the same supply voltage as the 80C51BH. At 5V, the transducer draws less than 5 mA (Reference 7). It can normally be connected directly to one of the C51BH's port pins, as shown in Figure 22.

FREQUENCY MEASUREMENTS

Measuring a frequency means counting pulses for a known sample time. Two timer/counters can be used, one to mark off the sample time and one to count pulses. If the frequency being counted doesn't exceed 50 kHz or so, one may equally well connect the transducer signal to one of the external interrupt pins, and count pulses in software. That frees up one timer, with very little cost in CPU time.

The count that is directly obtained is TxF, where T is the sample time and F is the frequency. The full scale

UPDATE_LCD:			
CLR	ET1		; Disable LCD drive interrupt.
MOV	DPTR, #TABLE_ADDRESS		; Look-up table begins at TABLE_ADDRESS
MOV	A, BCD_VALUE		; Digits to be displayed.
SHAP	A		; Move tens digit to low nibble.
ANL	A, #0FH		; Mask off high nibble.
MOV	A, EA+DPTR		; Tens digit pattern to accumulator.
MOV	TENS_DIGIT, A		; Update LCD tens digit.
MOV	A, BCD_VALUE		; Digits to be displayed.
ANL	A, #0FH		; Mask off tens digit.
MOV	A, EA+DPTR		; Ones digit pattern to accumulator.
MOV	C, DECIMAL_POINT		; Add decimal point to segment
MOV	ACC, 7, C		; pattern. Update LCD decimal point
MOV	ONES_DIGIT, A		; and ones digit.
SETB	ET1		; Re-enable LCD drive interrupt.
RET			

270068-21

Figure 20. UPDATE_LCD Routine Writes Two Digits to an LCD

range is $T_x(F_{max}-F_{min})$. For n-bit resolution

$$1 \text{ LSB} = \frac{T_x(F_{max}-F_{min})}{2^n}$$

Therefore the sample time required for n-bit resolution is

For example, 8-bit resolution in the measurement of a frequency that varies between 7 kHz and 9 kHz would require, according to this formula, a sample time of 128 ms. The maximum acceptable frequency count would be $128 \text{ ms} \times 9 \text{ kHz} = 1152$ counts. The minimum would be 896 counts. Subtracting 896 from each frequency count (or presetting the frequency counter to $= 896 = 0FC80H$) would allow the frequency to be reported on a scale of 0 to FF in hex digits.

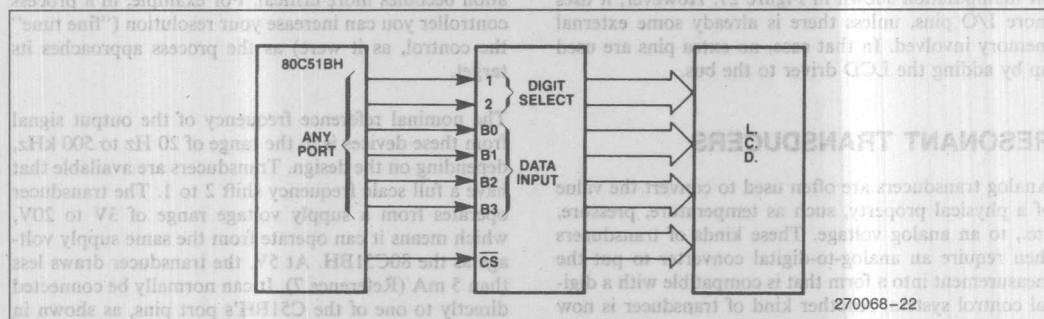


Figure 21a. Using an LCD Driver

```

UPDATE_LCD:
MOV     A, DISPLAY_HI      ; High byte of 4-digit display.
DIBIT_SELECT_2             ; Select leftmost digit of LCD.
SETB    DIBIT_SELECT_1     ; (Digit address = 11B.)
CALL    SHIFT_AND_LOAD     ; High nibble of high byte to selected digit.
CLR     DIBIT_SELECT_1     ; Select second digit of LCD (address = 10B).
CALL    SHIFT_AND_LOAD     ; Low nibble of high byte to selected digit.
MOV     A, DISPLAY_LO      ; Low byte of 4-digit display.
CLR     DIBIT_SELECT_2     ; Select third digit of LCD.
SETB    DIBIT_SELECT_1     ; (Digit address = 01B.)
CALL    SHIFT_AND_LOAD     ; High nibble of low byte to selected digit.
CLR     DIBIT_SELECT_1     ; Select fourth digit (address = 00B).
CALL    SHIFT_AND_LOAD     ; Low nibble of low byte to selected digit.
RET

SHIFT_AND_LOAD:
RLC     A                   ; MSB to carry bit (CY).
MOV     DATA_INPUT_B3,C   ; CY to Data Input pin B3.
RLC     A                   ; Next bit to CY.
MOV     DATA_INPUT_B2,C   ; CY to Data Input pin B2.
RLC     A                   ; Next bit to CY.
MOV     DATA_INPUT_B1,C   ; CY to Data Input pin B1.
RLC     A                   ; Last bit to CY.
MOV     DATA_INPUT_B0,C   ; CY to Data Input pin B0.
CLR     CHIP_SELECT        ; Toggle Chip Select.
SETB    CHIP_SELECT        ; 0-to-1 transition latches info.
RET
    
```

Figure 21b. UPDATE_LCD Routine Writes 4 Digits to an LCD Driver

```

UPDATE_LCD:
MOV     A, DISPLAY_HI      ; High byte of 4-digit display.
DIBIT_SELECT_2             ; Select leftmost digit of LCD.
SETB    DIBIT_SELECT_1     ; (Digit address = 11B.)
CALL    SHIFT_AND_LOAD     ; High nibble of high byte to selected digit.
CLR     DIBIT_SELECT_1     ; Select second digit of LCD (address = 10B).
CALL    SHIFT_AND_LOAD     ; Low nibble of high byte to selected digit.
MOV     A, DISPLAY_LO      ; Low byte of 4-digit display.
CLR     DIBIT_SELECT_2     ; Select third digit of LCD.
SETB    DIBIT_SELECT_1     ; (Digit address = 01B.)
CALL    SHIFT_AND_LOAD     ; High nibble of low byte to selected digit.
CLR     DIBIT_SELECT_1     ; Select fourth digit (address = 00B).
CALL    SHIFT_AND_LOAD     ; Low nibble of low byte to selected digit.
RET
    
```

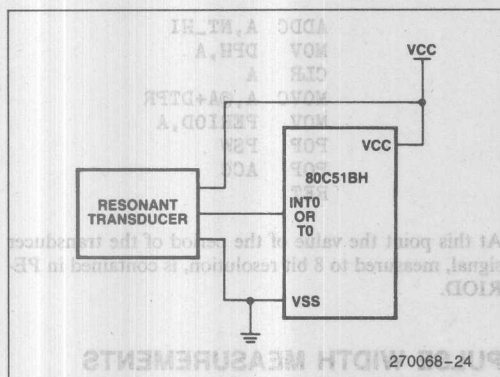


Figure 22. Resonant Transducer Does Not Require an A/D Converter

To implement the measurement, one timer is used to establish the sample time. The timer is preset to a value that causes it to roll over at the end of the sample time, generating an interrupt and waking the CPU from its Idle mode. The required preset value is the 2's complement negative of the sample time measured in machine cycles. The conversion from sample time to machine cycles is to multiply it by 1/12 the clock frequency. For example, if the clock frequency is 12 MHz, then a sample time of 128 ms is

$$(128 \text{ ms}) \times (12000 \text{ kHz})/12 = 128000 \text{ machine cycles.}$$

Then the required preset value to cause the timer to roll over in 128 ms is

$$-128000 = \text{FE0C00, in hex digits.}$$

Note that the preset value is 3 bytes wide whereas the timer is only 2 bytes wide. This means the timer must be augmented in software in the timer interrupt routine to three bytes. The 80C51BH has a DJNZ instruction (decrement and jump if not zero) that makes it easier to code the third timer byte to count down instead of up. If the third timer byte counts down, its reload value is the 2's complement of what it would be for an up-counter. For example, if the 2's complement of the sample time is FE0C00, then the reload value for the third timer byte would be 02, instead of FE. The timer interrupt routine might then be:

```
TIMER_INTERRUPT_ROUTINE:
    DJNZ THIRD_TIMER_BYTE,OUT
    MOV TLO,#0
    MOV THO,#0CH
    MOV THIRD_TIMERBYTE,#2
    MOV FREQUENCY_COUNTER_LO
;Preset COUNTER to -896:
    MOV COUNTER_LO,#80H
    MOV COUNTER_HI,#0FCH
OUT: RETI
```

At this point the value of the frequency of the transducer signal, measured to 8 bit resolution, is contained in FREQUENCY. Note that the timer can be reloaded on the fly. Note too that the timer can be reloaded on the fly. Note too that for 8-bit resolution only the low byte of the frequency counter needs to be read, since the high byte is necessarily 0. However, one may want to test the high byte to ensure that it is zero, as a sanity check on the data. Both bytes, of course must be reloaded.

PERIOD MEASUREMENTS

Measuring the period of the transducer signal means measuring the total elapsed time over a known number, N, of transducer pulses. The quantity that is directly measured is NT, where T is the period of the transducer signal in machine cycles. The relationship between T in machine cycles and the transducer frequency F in arbitrary frequency units is

$$T = \frac{\text{Fxtal}}{F} \times (1/12),$$

where Fxtal is the 80C51BH clock frequency, in the same units as F.

The full scale range then is $N \times (T_{\text{max}} - T_{\text{min}})$. For n-bit resolution.

$$1 \text{ LSB} = \frac{Ns(T_{\text{max}} - T_{\text{min}})}{2^n}$$

Therefore the number of periods over which the elapsed time should be measured is

$$N = \frac{2^n}{T_{\text{max}} - T_{\text{min}}}$$

However, N must also be an integer. It is logical to evaluate the above formula (don't forget T_{max} and T_{min} have to be in machine cycles) and select for N the next higher integer. This selection gives a period measurement that has somewhat more than n-bit resolution, but it can be scaled back if desired.

For example, suppose we want 8-bit resolution in the measurement of the period of a signal whose frequency varies from 7.1 kHz to 9 kHz. If the clock frequency is 12 MHz, then T_{max} is (12000 kHz/7.1 kHz) x (1/12) = 141 machine cycles. T_{min} is 111 machine cycles. The required value for N, then, is 256/(141-111) = 8.53 periods, according to the formula. Using N = 9 periods will give a maximum NT value of 141 x 9 = 1269 machine cycles. The minimum NT will be 111 x 9 = 999 machine cycles. A lookup table can be used to

scale these values back to a range of 0 to 255, giving precisely the 8-bit resolution desired.

To implement the measurement, one timer is used to measure the elapsed time, NT. The transducer is connected to one of the external interrupt pins, and this interrupt is configured to the transition-activated mode. In the transition-activated mode every 1-to-0 transition in the transducer output will generate an interrupt. The interrupt routine counts transducer pulses, and when it gets to the predetermined N, it reads and clears the timer. For the specific example cited above, the interrupt routine might be:

```

INTERRUPT_RESPONSE:
    DJNZ N,OUT
    MOV N,#9
    CLR EA
    CLR TR1
    MOV NT_LO,TL1
    MOV NT_HI,TH1
    MOV TL1,#9
    MOV TH1,#0
    SETB TR1
    SETB EA
    CALL LOOKUP_TABLE
OUT:    RETI

```

In this routine a pulse counter N is decremented from its preset value, 9, to zero. When the counter gets to zero it is reloaded to 9. Then all interrupts are blocked for a short time while the timer is read and cleared. The timer is stopped during the read and clear operations, so "clearing" it actually means presetting it to 9, to make up for the 9 machine cycles that are missed while the timer is stopped.

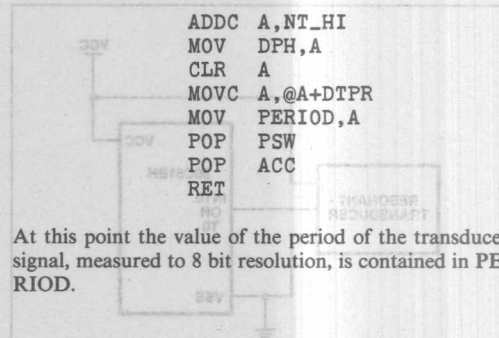
The subroutine LOOKUP_TABLE is used to scale the measurement back to the desired 8-bit resolution. It can also include built-in corrections for errors or nonlinearities in the transducer's transfer function.

The subroutine uses the MOVCA, @A + DPTR instruction to access the table, which contains 270 entries commencing at the 16-bit address referred to as TABLE. The subroutine must compute the address of the table entry that corresponds to the measured value of NT. This address is

```

DPTR = TABL + NT - NTMIN,
where NTMIN = 999, in this specific example.
LOOKUP_TABLE:
    PUSH ACC
    PUSH PSW
    MOV A,#LOW(TABLE-NTMIN)
    ADD A,NT_LO
    MOV DPL,A
    MOV A,#HIGH(TABLE-NTMIN)

```



At this point the value of the period of the transducer signal, measured to 8 bit resolution, is contained in PERIOD.

PULSE WIDTH MEASUREMENTS

The 80C51BH timers have an operating mode which is particularly suited to pulse width measurements, and will be useful in these applications if the transducer signal has a fixed duty cycle.

In this mode the timer is turned on by the on-chip circuitry in response to an input high at the external interrupt pin, and off by an input low, and it can do this while the 80C51BH is in Idle. (The "GATE" mode of timer operation is described in the Intel Microcontroller Handbook.) The external interrupt itself can be enabled, so the same 1-to-0 transition from the transducer that turns off the timer also generates an interrupt. The interrupt routine then reads and resets the timer.

The advantage of this method is that the transducer signal has direct access to the timer gate, with the result that variations in interrupt response time have no effect on the measurement.

Resonant transducers that are designed to fully exploit the GATE mode have an internal divide-by-N circuit that fixes the duty cycle at 50% and lowers the output frequency to the range of 250 to 500 Hz (to control RFI). The transfer function between transducer period and measurand is approximately linear, with known and repeatable error functions.

HMOS/CHMOS Interchangeability

The CHMOS version of the 8051 is architecturally identical with the HMOS version, but there are nevertheless some important differences between them which the designer should be aware of. In addition, some applications require interchangeability between HMOS and CHMOS parts. The differences that need to be considered are as follows:

External Clock Drive: To drive the HMOS 8051 with an external clock signal, one normally grounds the XTAL1 pin and drives the XTAL2 pin. To drive the CHMOS 8051 with an external clock signal, one must drive the XTAL1 pin and leave the XTAL2 pin unconnected. The reason for the difference is that in the

HMOS 8051, it is the XTAL2 pin that drives the internal clocking circuits, whereas in the CHMOS version it is the XTAL1 pin that drives the internal clocking circuits.

There are several ways to design an external clock drive to work with both types. For low clock frequencies (below 6 MHz), the HMOS 8051 can be driven in the same way as the CHMOS version, namely, through XTAL1 with XTAL2 unconnected. Another way is to drive both XTAL1 and XTAL2; that is, drive XTAL1 and use an external inverter to derive from XTAL1 a signal with which to drive XTAL2.

In either case, a 74HC or 74HCT circuit makes an excellent driver for XTAL1 and/or XTAL2, because neither the HMOS nor the CHMOS XTAL pins have TTL-like input logic levels.

Unused Pins: Unused pins of Ports 1, 2 and 3 can be ignored in both HMOS and CHMOS designs. The internal pullups will put them into a defined state. Unused Port 0 pins in 8051 applications can be ignored, even if they're floating. But in 80C51BH applications, these pins should not be left afloat. They can be externally pulled up or down, or they can be internally pulled down by writing 0s to them.

8031/80C31BH designs may or may not need pullups on Port 0. Pullups aren't needed for program fetches, because in bus operations the pins are actively pulled high or low by either the 8031 or the external program memory. But they are needed for the CHMOS part if the Idle or Power Down mode is invoked, because in these modes Port 0 floats.

Logic Levels: If V_{CC} is between 4.5V and 5.5V, an input signal that meets the HMOS 8051's input logic levels will also meet the CHMOS 80C51BH's input logic levels (except for XTAL1/XTAL2 and RST). For the same V_{CC} condition, the CHMOS device will reach or surpass the output logic levels of the HMOS device. The HMOS device will not necessarily reach the output logic levels of the CHMOS device. This is an important consideration if HMOS/CHMOS interchangeability must be maintained in an otherwise CMOS system.

HMOS 8051 outputs that have internal pullups (Ports 1, 2, and 3) "typically" reach 4V or more if I_{OH} is zero, but not fast enough to meet timing specs. Adding an external pullup resistor will ensure the logic level, but still not the timing, as shown in Figure 23. If timing is an issue, the best way to interface HMOS to CMOS is through a 74HCT circuit.

Idle and Power Down: The Idle and Power Down modes exist only on the CHMOS devices, but if one

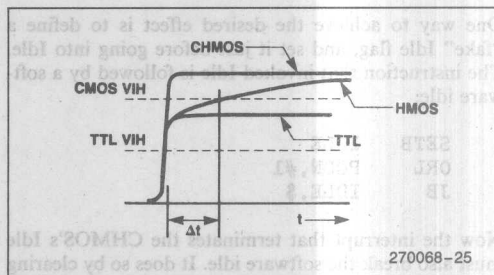


Figure 23. 0-to-1 Transition Shows Unspec'd Delay (Δt) in HMOS to 74HC Logic

wishes to preserve the capability of interchanging HMOS and CHMOS 8051s the software has to be designed so that the HMOS parts will respond in an acceptable manner when a CHMOS reduced power mode is invoked.

For example, an instruction that invokes Power Down can be followed by a "JMP \$":

```
CLR    EA
ORL    PCON, #2
JMP    $
```

The CHMOS and HMOS parts will respond to this sequence of code differently. The CHMOS part, going into a normal CHMOS Power Down Mode, will stop fetching instructions until it gets a hardware reset. The HMOS part will go through the motions of executing the ORL instruction, and then fetch the JMP instruction. It will continue fetching and executing JMP \$ until hardware reset.

Maintaining HMOS/CHMOS 8051 interchangeability in response to Idle requires more planning. The HMOS part will not respond to the instruction that puts the CHMOS part into Idle, so that instruction needs to be followed by a software idle. This would be an idling loop which would be terminated by the same conditions that would terminate the CHMOS's hardware Idle. Then when the CHMOS device goes into Idle, the HMOS version executes the idling loop, until either a hardware reset or an enabled interrupt is received. Now if Idle is terminated by an interrupt, execution for the CHMOS device will proceed after RETI from the instruction following the one that invoked Idle. The instruction following the one that invoked Idle is the idling loop that was inserted for the HMOS device. At this point, both the HMOS and CHMOS devices must be able to fall through the loop to continue execution.

One way to achieve the desired effect is to define a "fake" Idle flag, and set it just before going into Idle. The instruction that invoked Idle is followed by a software idle:

```
SETB   IDLE
ORL     PCON, #1
JB      IDLE, $
```

Now the interrupt that terminates the CHMOS's Idle must also break the software idle. It does so by clearing the "Idle" bit:

```
CLR     IDLE
RETI
```

Note too that the PCON register in the HMOS 8051 contains only one bit, SMOD, whereas the PCON register in CHMOS contains SMOD plus four other bits. Two of those other bits are general purpose flags. Maintaining HMOS/CHMOS interchangeability requires that these flags not be used.

REFERENCES

1. Pawlowski, Moroyan, Alnether, "Inside CMOS Technology," *BYTE magazine*, Sept., 1983. Available as Article Reprint AR-302.
2. Kokkonen, Pashley, "Modular Approach to C-MOS Technology Tailors Process to Application," *Electronics*, May, 1984. Available as Article Reprint AR-332.
3. Williamson, T., *Designing Microcontroller Systems for Electrically Noisy Environments*, Intel Application Note AP-125, Feb. 1982.
4. Williamson, T., "PC Layout Techniques for Minimizing Noise," *Mini-Micro Southeast*, Session 9, Jan., 1984.
5. Alnether, J., *High Speed Memory System Design Using 2147H*, Intel Application Note AP-74, March 1980.
6. Ott, H., "Digital Circuit Grounding and Interconnection," *Proceedings of the IEEE Symposium on Electromagnetic Compatibility*, pp. 292-297, Aug. 1981.
7. *Digital Sensors by Technar*, Technar Inc., 205 North 2nd Ave., Arcadia, CA 91006.

Enhanced Serial Port on the 83C51FA

RECOGNITION

Automatic Address Recognition reduces the CPU time required to service the serial port. Since the CPU is only interrupted when it receives its own address, the software overhead to compare addresses is eliminated. This would also effectively reduce the sophistication of the serial protocol when numerous controllers are sharing the same serial link.

This same feature can also be used in conjunction with the Idle Mode to reduce the system's overall power consumption. For instance, a master may need to communicate with only one slave at a time. With all slaves in Idle Mode, only that one slave would be interrupted to respond to the master's transmission. When a slave responds, each slave would have to "wake up," check for its address. Limiting the interrupt to check for its address, thus reduces the amount of current drawn by the system and thus reduces the power consumption.

In multiprocessor applications the serial port is configured in either of the 9-bit modes (Mode 2 or 3). Mode 2 has a fixed baud rate whereas Mode 3 is variable. For more information on the different serial port modes refer to the "Serial Port 2 or 3" section.

Automatic Address Recognition is enabled by setting the SADR bit in the 83C51FA's SADR register. The SADR bit = 1. The Receiver Interrupt flag will get set when the received byte matches the SADR. Given or Broadcast Address. The SADR bit = 0 to enable reception of data bytes (9th bit = 0) from the master.

The master can selectively communicate with groups of slaves by using the Given Address. Addressing all slaves at once is possible with the Broadcast Address. These addresses are defined for each slave in two new Special Function Registers: SADDR and SADEN.

A slave's individual address is specified in SADDR. SADEN is a mask byte that defines don't-care bits for the Given Address. These don't-care bits allow flexibility in the user-defined protocol to address one or more slaves. The following is an example of how to define Given Address and selectively address different slaves.

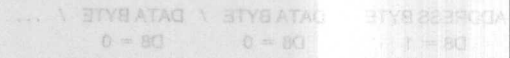
83C51FA with the addition of two new features: Auto-made Address Recognition and Framing Error Detection. Automatic Address Recognition facilitates multiprocessor communication by reducing CPU overhead. Framing Error Detection increases communication reliability by checking each reception for a valid stop bit.

This Application Note explains how to use these new features with samples of code for typical applications. A section is also included which reviews how to set up the serial port for multiprocessor applications.

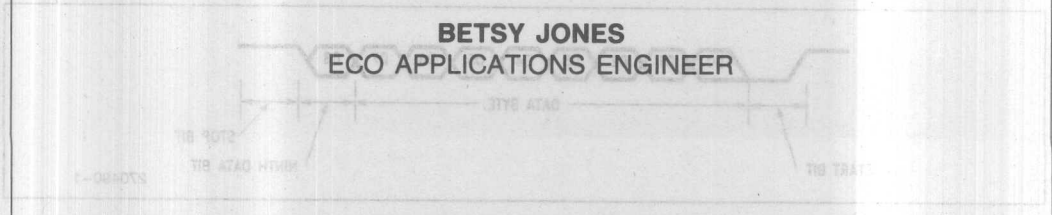
In applications where multiple controllers jointly perform a task, the master controller must be able to communicate with individual slaves. To do this, the master first identifies the target slave (or slaves) with an address and then transmits a block of data. The target slave must be able to identify their own address before receiving any data bytes.

The serial port on the 83C51 provides a 9-bit mode to facilitate multiprocessor communication. The 9th bit allows the controller to distinguish between address and data bytes. In this mode a total of 11 bits are received or transmitted for each byte. Figure 1 shows a programable data stream.

The 9th bit is set to 1 to identify address bytes. A typical data stream is shown below:



Initially the slave is setup to only receive address bytes. Once it receives its own address, the slave recognizes itself to receive data. On the 83C51 serial port, an address byte identifies all slaves for an address comparison. On the 83C51FA, however, Automatic Address Recognition allows the addressed slave to be the only one interrupted. In this address comparison occurs in hardware, not software. With this feature, the master controller can establish communication with one or more slaves without all the slaves having to respond to the transmission.



The serial port on the 8051 has been enhanced on the 83C51FA with the addition of two new features: Automatic Address Recognition and Framing Error Detection. **Automatic Address Recognition** facilitates multiprocessor communications by reducing CPU overhead. **Framing Error Detection** increases communication reliability by checking each reception for a valid stop bit.

This Application Note explains how to use these new features with samples of code for typical applications. A section is also included which reviews how to set up the serial port for multiprocessor applications.

MULTIPROCESSOR COMMUNICATIONS

In applications where multiple controllers jointly perform a task, the master controller must be able to communicate selectively with individual slaves. To do this, the master first identifies the target slave (or slaves) with an address byte and then transmits a block of data. The target slaves must be able to identify their own address before receiving any data bytes.

The serial port on the 8051 provides a 9-bit mode to facilitate multiprocessor communication. The 9th bit allows the controller to distinguish between address and data bytes. In this mode, a total of 11 bits are received or transmitted: a start bit (0), 8 data bits (LSB first), a programmable 9th bit, and a stop bit (1). See Figure below.

The 9th bit is set to 1 to identify address bytes and set to 0 for data bytes. A typical data stream is seen below:

ADDRESS BYTE /	DATA BYTE /	DATA BYTE /	...
D8 = 1	D8 = 0	D8 = 0	

Initially the slave is set up to only receive address bytes. Once it receives its own address, the slave reconfigures itself to receive data. On the 8051 serial port, an address byte interrupts **all** slaves for an address comparison. On the 83C51FA, however, Automatic Address Recognition allows the addressed slave to be the **only** one interrupted; that is, the address comparison occurs in hardware, not software. With this feature, the master controller can establish communication with one or more slaves without all the slaves having to respond to the transmission.

AUTOMATIC ADDRESS RECOGNITION

Automatic Address Recognition reduces the CPU time required to service the serial port. Since the CPU is only interrupted when it receives its own address, the software overhead to compare addresses is eliminated. This would also effectively reduce the sophistication of the serial protocol when numerous controllers are sharing the same serial link.

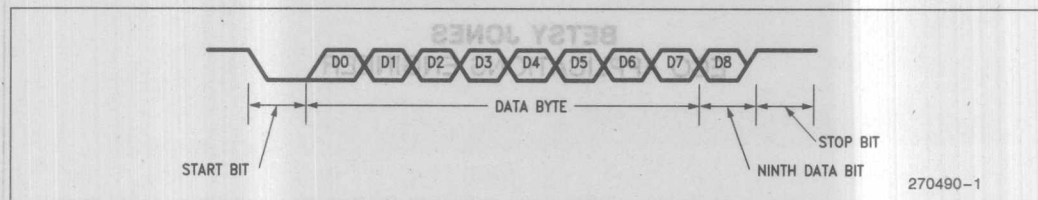
This same feature can also be used in conjunction with the Idle Mode to reduce the system's overall power consumption. For instance, a master may need to communicate with only one slave at a time. With all slaves in Idle Mode, only that one slave would be interrupted to respond to the master's transmission. Without Automatic Addressing, each slave would have to "wake up" to check for its address. Limiting the interruptions reduces the amount of current drawn by the system and thus reduces the power consumption.

In multiprocessor applications the serial port is configured in either of the 9-bit modes (Mode 2 or 3). Mode 2 has a fixed baud rate whereas Mode 3 is variable. For more information on the different serial port modes refer to the "Serial Port Set Up" section.

Automatic Address Recognition is enabled by setting the SM2 bit in SCON. Each slave has its SM2 bit set waiting for an address byte (9th bit = 1). The Receive Interrupt (RI) flag will get set when the received byte corresponds to either a Given or Broadcast Address. The slave then clears its SM2 bit to enable reception of data bytes (9th bit = 0) from the master.

The master can selectively communicate with groups of slaves by using the Given Address. Addressing all slaves at once is possible with the Broadcast Address. These addresses are defined for each slave by two new Special Function Registers: SADDR and SADEN.

A slave's individual address is specified in SADDR. SADEN is a mask byte that defines don't-cares to form the Given Address. These don't-cares allow flexibility in the user-defined protocol to address one or more slaves. The following is an example of how to define Given Addresses and selectively address different slaves.



Slave 1			
SADDR	=	1111	0001
SADEN	=	1111	1010
GIVEN	=	1111	0X0X

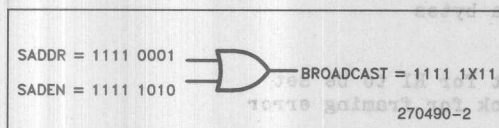
Slave 2			
SADDR	=	1111	0011
SADEN	=	1111	1001
GIVEN	=	1111	0XX1

The SADEN bytes have been selected such that bit 1 (LSB) is a don't-care for Slave 1's Given Address, but bit 1 = 1 for Slave 2. Thus, to selectively communicate with just Slave 1 an address with bit 1 = 0 would be used (e.g. 1111 0000).

Similarly, bit 2 = 0 for Slave 1, but is a don't-care for Slave 2. Now to communicate with just Slave 2 an address with bit 2 = 1 would be used (e.g. 1111 0111).

Finally, to communicate with both slaves at once the address must have bit 1 = 1 and bit 2 = 0. Notice, however, that bit 3 is a "don't-care" for both slaves. This allows two different addresses to select both slaves (1111 0001 or 1111 0101). If a third slave was added that required its bit 3 = 0, then the latter address could be used to communicate with Slave 1 and 2 but not Slave 3.

The master can also communicate with all slaves at once with the Broadcast Address. It is formed from the logical OR of SADDR and SADEN with zeros defined as don't-cares. For example, the Broadcast address for Slave 1 would be formed as follows:



The don't-cares also allow flexibility in defining the Broadcast Address, but in most applications a Broadcast Address will be OFFH.

SADDR and SADEN are located at address A9H and B9H, respectively. On Reset, SADDR and SADEN are initialized to 00H which defines the Given and Broadcast Addresses as XXXX XXXX (all don't-cares). This assures the 83C51FA serial port to be backwards compatible with the other MCS[®]-51 products which do not implement Automatic Addressing.

FRAMING ERROR DETECTION

Framing Error Detection is another new feature on 83C51FA serial port which allows the receiving controller to check for valid stop bits in Modes 1, 2, or 3. A missing stop bit can be caused, for example, by noise on the serial lines or transmission by two CPUs simultaneously.

If a stop bit is missing a Framing Error bit FE will be set. This bit can then be checked in software after each reception to detect communication errors. Once set, the FE bit must be cleared in software. A valid stop bit will not clear FE.

The FE bit is located in SCON and shares the same bit address as SM0. To determine which is accessed, a new control bit called SMOD0 has been added in the PCON register (see figures below). If SMOD0 = 0, then accesses to SCON.7 are to SM0. If SMOD0 = 1, then accesses to SCON.7 are to FE.

PCON: Power Control Register (Not Bit Addressable)

SMOD1	SMOD0	—	POF	GF1	GF0	PD	IDL
-------	-------	---	-----	-----	-----	----	-----

Address = 87H

SCON: Serial Port Control Register (Bit Addressable)

SM0/FE	SM1	SM2	REN	TB8	RB8	TI	RI
--------	-----	-----	-----	-----	-----	----	----

Address = 98H

SERIAL PORT SOFTWARE

The following sections of code show examples of how to invoke Automatic Addressing and Framing Error Detection. Routines for both the slave and master are given. Code is also included to initialize both serial ports; however, for more information on setting up the serial port refer to the next section.

For this example, the master and slave are transmitting/receiving at 9600 baud with a 12 MHz crystal frequency. To obtain this baud rate, the serial port is configured in Mode 3 and Timer 2 is used as the baud-rate generator.

Listing 1 shows the initialization for the slave. Notice that Automatic Addressing and Framing Error Detection are enabled. The Given and Broadcast addresses for this slave are taken from Slave 1 in the previous example. A temporary byte has also been defined to store the incoming data byte.

The slave will remain in Idle Mode until it is interrupted by its own address. At that point, it clears the SM2

Listing 1. Initialization Routine for the Slave

```

ORG 00H                                ; Program Error Detection is another
LJMP INIT                                ; 83C31FA serial port which allows
                                        ; master to check for valid stop bits in Mode 1, 2, or 3. A
                                        ; missing stop bit can be caused, for example, by
                                        ; the serial line being busy.
ORG 0023H                                ;
LJMP SERIAL_PORT_INTERRUPT              ;

TEMP DATA 30H                          ; Temporary storage byte

INIT: MOV SCON, #0FOH                    ; Mode 3, enable Auto Addressing
                                        ; and reception
ORL PCON, #40H                          ; FE bit accessed (SMOD0 = 1)
MOV RCAP2H, #0FFH                       ; Reload values for 9600 Baud
MOV RCAP2L, #0D9H                       ;
MOV T2CON, #34H                         ; Timer 2 set up, TR2 = 1 turns
                                        ; timer on

INTERRUPTS: SETB EA                      ; Enable global interrupt
SETB ES                                  ; Enable serial port interrupt

ADDRESSES: MOV SADDR, # 11110001         ; Define Given & Broadcast
MOV SADEN, # 11111010                  ; Addresses
                                        ; GIVEN = 11110X0X
                                        ; BROADCAST = 11111X11

IDLE_MODE: ORL PCON, #01H                ; Invoke Idle Mode

```

Listing 2. Receive Routine for the Slave

```

SERIAL_PORT_INTERRUPT:
    PUSH PSW
    CLR RI                                ; RI set when address is
                                        ; recognized & must be cleared
                                        ; in software
    CLR SM2                              ; Reconfigure slave to receive
                                        ; data bytes

RECEIVE_DATA:
    JNB RI, $                            ; Wait for RI to be set
    MOV C, SCON.7                        ; Check for framing error
    JC FRAMING_ERROR
    MOV TEMP, SBUF                       ; Receive data byte & store
                                        ; in temporary location
    CLR RI                                ; Clear flag for next
                                        ; reception
    SETB SM2                             ; Re-enable Automatic
                                        ; Addressing

    POP PSW
    RETI

FRAMING_ERROR:
    CLR SCON.7                           ; Clear FE bit
    CLR C
    .
    .
    .
    POP PSW
    RETI

```

Listing 3. Initialization and Transmit Routines for the Master

```

GIVEN_1      equ      11110001B
MESSAGE_1    data     30H

INIT:  MOV  SCON, #0DOH      ; Mode 3, REN = 1
       MOV  RCAP2H, #OFFH    ; 9600 Baud
       MOV  RCAP2L, #0D9H    ; Automatic Addressing (SM2 = 1)
       MOV  T2CON, #34H     ; Timer 2 set up, TR2 = 1

TRANSMIT_ADDRESS:
  CLR  TI
  SETB TB8      ; Mark 1st byte as an address
               ; byte (9th bit = 1)
  MOV  SBUF, #GIVEN_1    ; Send address
  JNB  TI, $          ; Wait for transmission
               ; complete
  CLR  TI
               ; Clear flag for next
               ; transmission

TRANSMIT_DATA:
  CLR  TB8      ; Mark 2nd byte as a data
               ; byte (9th bit = 0)
  MOV  SBUF, MESSAGE_1    ; Send data byte
  JNB  TI, $
  CLR  TI

```

bit to enable reception of data bytes. Depending on the user's protocol, more than one data byte may actually be received. This example, however, assumes only one byte of data follows each address byte.

Listing 2 shows the receive routine. Notice that when the data byte is received, the software checks for a framing error. The error routine could, for example, send an error message to the master and ask the master to re-transmit the last message. Before exiting the routine the SM2 is set to 1 to reenables Automatic Addressing. Once the slave has responded to the master's command, it could also put itself back into Idle Mode to wait for the next message.

The initialization routine for the master in Listing 3 is very similar to the slave. In this example, however, the master does not need Automatic Addressing; it is simply transmitting address and data bytes. GIVEN_1 is a byte to address the slave in the above example. MESSAGE_1 is a register that contains the data byte sent to this slave. Its value is arbitrary for the sample code.

SERIAL PORT SET UP

This section describes how to initialize the 83C51FA serial port for multiprocessor applications. Two different modes are available which provide 9-bit operation:

Mode 2 which has a fixed baud rate and Mode 3 which has a variable baud rate. Baud rates can be generated by either Timer 1 or Timer 2 (available on the 83C51FA but not the 8051). Deciding which mode and timer to use is determined by the desired baud rate and clock frequency of the particular application.

Another consideration is the tolerance needed between serial ports. Since the serial port re-synchs its receiver at every start bit, only 8 or 9 bit-times are available to accumulate timing errors. As a result, the receiver and transmitter only have to be within about 5% of each other's baud rate. Allowing equal error to both transmitter and receiver, only about 2% accuracy is actually needed.

Following is a discussion of both Modes 2 and 3 and examples of how to program each. The mode selection bits (SM0 and SM1) are located in SCON. The REN bit must also be set to enable reception.

SCON: Serial Port Control Register (Bit Addressable)

SM0	SM1	SM2	REN	TB8	RB8	TI	RI
-----	-----	-----	-----	-----	-----	----	----

Address = 98H

Mode	SM0	SM1	Baud Rate
2	0	0	Fosc/64 or Fosc/32
3	1	1	Variable

Example 1. Serial Port Mode 2

; Frequency	= 12 MHz
; Desired Baud Rate	= 375 kBaud
;	= 1/32 (Osc Freq)
MOV SCON, #0B0H	; Serial port Mode 2
	; Automatic Addressing (SM2 = 1)
	; reception enabled (REN = 1)
ORL PCON, #80H	; SMOD1 = 1 to double baud rate

Mode 2

Mode 2 uses a fixed baud rate of 1/32 or 1/64 of the oscillator frequency depending on the value of the SMOD1 bit in PCON. This mode basically offers a choice of two high-speed baud rates. With a 12 MHz clock frequency, baud rates of 187.5 kbaud or 375 kbaud can be obtained.

None of the timer/counters need to be set up for Mode 2. Only the SFRs SCON and PCON need to be defined.

PCON: Power Control Register (Not Bit Addressable)

SMOD1	SMOD0	—	POF	GF1	GF0	PD	IDL
-------	-------	---	-----	-----	-----	----	-----

Address = 87H

The baud rate in this mode is calculated by:

$$\text{Mode 2 Baud Rate} = \frac{2^{\text{SMOD1}} \times \text{Osc Freq}}{64}$$

SMOD1 = 0, Baud Rate = 1/64 Osc Freq

SMOD1 = 1, Baud Rate = 1/32 Osc Freq

Mode 3

Mode 3 of the serial port has a variable baud rate generated by either Timer 1 or Timer 2. The baud rate is generated by the rollover rate of the selected timer. The timer is operated in an auto-reload mode so it will roll over to the reload value selected in software.

Baud rates based off Timer 2 have less granularity so that almost any baud rate can be obtained at a given clock frequency. However, Timer 1 is sufficient if the desired baud rate can be obtained at the specified clock frequency. Remember baud rates only need about 2% accuracy.

Timer 1 Set Up

To generate baud rates Timer 1 is usually configured in 8-bit auto-reload mode (Mode 2). The mode select bits

are M1 and M0 located in TMOD. To turn on Timer 1 the TR1 bit in TCON must be set. Also, the Timer 1 interrupt should be disabled in this application so that when the timer overflows it does not generate an interrupt.

TMOD: Timer/Counter Mode Control Register (Not bit addressable)

GATE	C/T	M1	M0	GATE	C/T	M1	M0
------	-----	----	----	------	-----	----	----

Timer 1

Timer 0

Address = 89H

TCON: Timer/Counter Control Register (Bit addressable)

TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0
-----	-----	-----	-----	-----	-----	-----	-----

Address = 88H

The formula for calculating the baud rate is given below. TH1 is the reload value for Timer 1 when it overflows.

$$\text{Baud Rate} = \frac{K \times \text{Osc Freq}}{32 \times 12 \times [256 - (\text{TH1})]}$$

K = 1 if SMOD1 = 0.

K = 2 if SMOD1 = 1. (SMOD1 is at PCON.7)

If the baud rate is known, the reload value TH1 can be calculated by:

$$\text{TH1} = 256 - \frac{K \times \text{Osc Freq}}{384 \times \text{Baud Rate}}$$

TH1 must be an integer value. Rounding off TH1 to the nearest integer may not produce the desired baud rate with the 2% accuracy required. In this case, another crystal frequency may have to be chosen.

Refer to Table 1 for timer reload values for commonly used baud rates.

Table 1. Commonly Used Baud Rates Generated by Timer 1

Baud Rate	Osc Freq	SMOD1	Timer 1	
			TMOD	Reload Value
62.5K	12 MHz	1	20	FFH
19.2K	11.06 MHz	1	20	FDH
9.6K	11.06 MHz	0	20	FDH
4.8K	11.06 MHz	0	20	FAH
2.4K	11.06 MHz	0	20	F4H
1.2K	11.06 MHz	0	20	E8H
300	6 MHz	0	20	CCH
110	6 MHz	0	20	72H

Example 2. Serial Port Mode 3, with Timer 1 as Baud-Rate Generator

; Frequency = 11.0 MHz	
; Desired Baud Rate = 19.2 kBaud	
; TH1 = 256 - $\frac{(2) \times (11.0 \times 10^6)}{(32) \times (12) \times (19200)}$	
; = 253 = FDH	
MOV SCON, #0FOH	; Serial port Mode 3, SM2 = 1,
ORL PCON, #80H	; REN = 1
MOV TMOD, #20H	; SMOD1 = 1
MOV TH1, #0FDH	; Timer 1 Mode 2
SETB TR1	; Reload value for desired baud
	; rate
	; Turn on Timer 1

It can be seen that the exact frequency to generate the standard baud rates (19.2K, 9600, 4800, etc.) is 11.06 MHz. However, it is not necessary to use this exact frequency. With a 2% tolerance any crystal value from 10.8 MHz to 11.3 MHz is sufficient.

Timer 2 Set Up

Timer 2 has a special baud-rate generator mode which transmits and receives at the same baud rate. This mode is invoked by setting both the RCLK and TCLK bits in T2CON. To turn Timer 2 on the TR2 bit should also be set.

Unlike Timer 1, this mode does not require that the timer overflow interrupt be disabled. That is, when Timer 2 is in the baud-rate generator mode, its interrupt is disconnected from the Timer 2 overflow. This

interrupt then becomes available as a third external interrupt. (For more information on external interrupts, refer to the chapter "Hardware Description of the 8051" in the Embedded Controller Handbook.)

T2CON: Timer/Counter 2 Control Register
(Bit Addressable)

TF2	EXF2	RCLK	TCLK	EXEN2	TR2	C/T2	CP/RL2
-----	------	------	------	-------	-----	------	--------

Address = C8H

This formula for calculating the baud rate is given below. (RCAP2H, RCAP2L) is the 16-bit reload value when Timer 2 overflows.

$$\text{Baud Rate} = \frac{\text{Osc Freq}}{32 \times [65536 - (\text{RCAP2H}, \text{RCAP2L})]}$$

where (RCAP2H, RCAP2L) is a 16-bit unsigned integer.

To obtain the reload value for RCAP2H and RCAP2L the above equation can be rewritten as:

$$(RCAP2H, RCAP2L) = 65536 - \frac{\text{Osc Freq}}{32 \times \text{Baud Rate}}$$

Refer to Table 2 for reload values for commonly used baud rates.

Notice that when using Timer 2, most standard baud rates can be obtained at 12 MHz.

**Table 2. Commonly Used Baud Rates
Generated by Timer 2**

Baud Rate	Osc Freq	Timer 2	
		RCAP2H	RCAP2L
375K	12 MHz	FF	FF
9.6K	12 MHz	FF	D9
4.8K	12 MHz	FF	B2
2.4K	12 MHz	FF	64
1.2K	12 MHz	FE	C8
300	12 MHz	FB	1E
110	12 MHz	F2	AF
300	6 MHz	FD	8F
110	6 MHz	F9	57

Example 3. Serial Port Timer with Timer 2 as Baud-Rate Generator

```

; Frequency          = 12 MHz
; Desired Baud Rate  = 9600 Baud
;
; (RCAP2H, RCAP2L) = 65536 - (12 x 106) / (32 x (9600))
;
; = 65497 = FFD9H
MOV SCON, #0F0H      ; Serial port Mode 3, SM2 = 1,
; REN = 1
MOV RCAP2H, #0FFH    ; Reload values for desired
MOV RCAP2L, #0D9H    ; baud rate
MOV T2CON, #34H      ; Timer 2 as baud rate
; generator, turn on Timer 2

```

interrupt then becomes available as a third external interrupt. (For more information on external interrupts, refer to the chapter "Hardware Description of the 8051" in the Embedded Controller Handbook.)

T2CON: Timer/Counter 2 Control Register
(8-bit Addressable)

Address = C8H

TF2	EXF2	RCLK	TCLK	EXEN2	TR2	C/T2	CP/RL2
-----	------	------	------	-------	-----	------	--------

This formula for calculating the baud rate is given below. (RCAP2H, RCAP2L) is the 16-bit reload value when Timer 2 overflows.

$$\text{Baud Rate} = \frac{\text{Osc Freq}}{32 \times (65536 - (RCAP2H, RCAP2L))}$$

where (RCAP2H, RCAP2L) is a 16-bit unsigned integer.

It can be seen that the exact frequency to generate the standard baud rates (19.2K, 9600, 4800, etc.) is 11.05 MHz. However, it is not necessary to use this exact frequency. With a 2% tolerance any crystal value from 10.8 MHz to 11.3 MHz is sufficient.

Timer 2 Set Up

Timer 2 has a special baud-rate generator mode which transmits and receives at the same baud rate. This mode is invoked by setting both the RCLK and TCLK bits in T2CON. To turn Timer 2 on the TR2 bit should also be set.

Unlike Timer 1, this mode does not require that the timer overflow interrupt be disabled. That is, when Timer 2 is in the baud-rate generator mode, its interrupt is disconnected from the Timer 2 overflow. This

BRIEF

(3) Receive routine.
(4) Transmit routine.

A complete listing of the routines and the test loop which was used to verify their operation is found in the Appendix. A total of three half-duplex channels were run at 2400 Baud in the test program. The listings shown here are simplified to one channel (Channel 0).

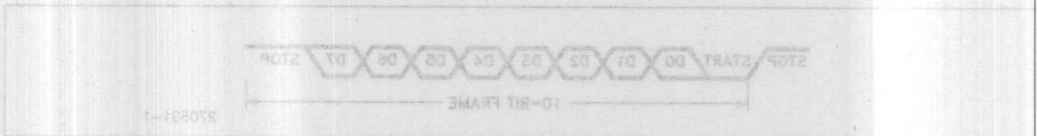
Variables

Listing 1 shows the variables used in both the receive and transmit routines. Flags are defined to signal the status of the reception or transmission of a byte (e.g. RCV_START_BIT, TXM_START_BIT). RCV_BUF and TXM_BUF simulate the on-chip serial port SBUF as two separate buffer registers. The temporary registers RCV_REG and TXM_REG are used to save bits as they are received or transmitted. Finally, two counter registers keep track of how many bits have been received or transmitted.

Variables are also needed to define one-half and one-full bit times in units of PCA timer ticks. (One bit time = 1 / baud rate). With the PCA timer incremented every machine cycle, the equation to calculate one bit time can be written as:

Software Serial Port Implemented with the PCA

The high and low byte of this value is placed in the variables FULL_HIGH and FULL_LOW respectively. HIGH is the value loaded into HALF_HIGH and HALF_LOW.



BETSY JONES
ECO APPLICATIONS ENGINEER

For microcontroller applications which require more than one serial port, the 83C21FA Programmable Counter Array (PCA) can implement additional half-duplex serial ports. If the on-chip UART is being used as an inter-processor link, the PCA can be used to interface the 83C21FA to additional asynchronous lines.

This application uses several different Compare/Counter modes available on the PCA to receive or transmit bytes of data. It is assumed the reader is familiar with the PCA and ASM51. For more information on the PCA, refer to the "Hardware Description of the 83C21FA" chapter in the Embedded Controller Handbook (Order No. 270531).

September 1988

The figure below shows the format of a standard 10-bit asynchronous frame. A 1 start bit (S), 8 data bits (D), and 1 stop bit (P). The start bit is used to synchronize the receiver to the transmitter at the leading edge of the start bit. The receiver must set up its timing logic to sample the incoming line in the center of each bit. Following the start bit are eight data bits which are transmitted least significant bit first. The stop bit is set to the opposite state of the start bit to guarantee that the leading edge of the stop bit will cause a transition on the line. It also provides a dead time on the line so that the receiver can maintain its synchronization.

The PCA can be configured to receive and transmit data using the Negative-Edge Capture mode. The software uses the PCA to detect the start bit. Then using the software timer mode, interrupts are generated to sample the incoming data bits. This same mode is used to clock out data when transmitting.

The Application Note contains four sections of code:

- (1) List of variables
- (2) Initialization routine

For microcontroller applications which require more than one serial port, the 83C51FA Programmable Counter Array (PCA) can implement additional half-duplex serial ports. If the on-chip UART is being used as an inter-processor link, the PCA can be used to interface the 83C51FA to additional asynchronous lines.

This application uses several different Compare/Capture modes available on the PCA to receive or transmit bytes of data. It is assumed the reader is familiar the PCA and ASM51. For more information on the PCA refer to the "Hardware Description of the 83C51FA" chapter in the Embedded Controller Handbook (Order No. 210918).

Introduction

The figure below shows the format of a standard 10-bit asynchronous frame: 1 start bit (0), 8 data bits, and 1 stop bit (1). The start bit is used to synchronize the receiver to the transmitter; at the leading edge of the start bit the receiver must set up its timing logic to sample the incoming line in the center of each bit. Following the start bit are eight data bits which are transmitted least significant bit first. The stop bit is set to the opposite state of the start bit to guarantee that the leading edge of the start bit will cause a transition on the line. It also provides a dead time on the line so that the receiver can maintain its synchronization.

Two of the Compare/Capture modes on the PCA are used in receiving and transmitting data bits. When receiving, the Negative-Edge Capture mode allows the PCA to detect the start bit. Then using the Software Timer mode, interrupts are generated to sample the incoming data bits. This same mode is used to clock out bits when transmitting.

This Application Note contains four sections of code:

- (1) List of variables
- (2) Initialization routine

- (3) Receive routine
- (4) Transmit routine.

A complete listing of the routines and the test loop which was used to verify their operation is found in the Appendix. A total of three half-duplex channels were run at 2400 Baud in the test program. The listings shown here are simplified to one channel (Channel 0).

Variables

Listing 1 shows the variables used in both the receive and transmit routines. Flags are defined to signify the status of the reception or transmission of a byte (e.g. RCV_START_BIT, TXM_START_BIT). RCV_BUF and TXM_BUF simulate the on-chip serial port SBUF as two separate buffer registers. The temporary registers, RCV_REG and TXM_REG, are used to save bits as they are received or transmitted. Finally, two counter registers keep track of how many bits have been received or transmitted.

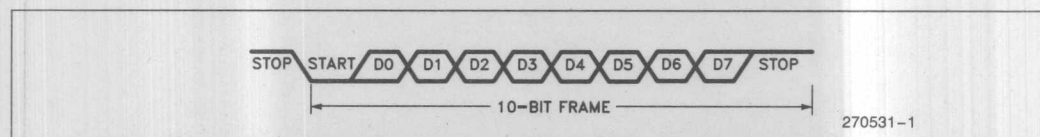
Variables are also needed to define one-half and one-full bit times in units of PCA timer ticks. (One bit time = $1 / \text{baud rate}$.) With the PCA timer incremented every machine cycle, the equation to calculate one bit time can be written as:

$$\frac{\text{Osc. Freq.}}{(12) \times (\text{baud rate})} = 1 \text{ bit time (in PCA timer ticks)}$$

In this example, the baud rate is 2400 at 16 MHz.

$$\frac{16 \text{ MHz}}{(12) \times (2400)} = 556 \text{ counts} = 22\text{C Hex}$$

The high and low byte of this value is placed in the variables FULL_BIT_HIGH and FULL_BIT_LOW, respectively. 115H is the value loaded into HALF_BIT_HIGH and HALF_BIT_LOW.



Listing 1. Variables used by the software serial port. Channel 0

; <u>Receive Routine</u>			
RCV_START_BIT_0	BIT	20H.0	; Indicates start bit ; has been received
RCV_DONE_0	BIT	20H.1	; Indicates data byte ; has been received
RCV_BUF_0	DATA	30H	; Software Receive ; "SBUF"
RCV_REG_0	DATA	31H	; Temporary register ; for receive bits
RCV_COUNT_0	DATA	32H	; Counter for receiving ; bits
; <u>Transmit Routine:</u>			
TXM_START_BIT_0	BIT	20H.3	; Indicates start bit ; has been transmitted
TXM_IN_PROGRESS_0	BIT	20H.4	; Indicates transmit is ; in progress
TXM_BUF_0	DATA	34H	; Software transmit ; "SBUF"
TXM_REG_0	DATA	35H	; Temporary register ; for transmitting bits
TXM_COUNT_0	DATA	36H	; Counter for transmit- ; ting bits
DATA_0	DATA	37H	; Register used for the ; test program
NEG_EDGE	EQU	11H	; Two modes of operation
S_W_TIMER	EQU	49H	; for compare/capture ; modules
HALF_BIT_HIGH	EQU	01H	; Half bit time = 115H
HALF_BIT_LOW	EQU	15H	
FULL_BIT_HIGH	EQU	02H	; Full bit time = 22CH
FULL_BIT_LOW	EQU	2CH	; 2400 Baud at 16 MHz

270531-4

Initialization

Listing 2 contains the initialization code for the receive and transmit process. Module 0 of the PCA is used as a receiver and is first set up to detect a negative edge from the start bit. Modules 2 and 3 are used for the additional 2 channels (see the Appendix). Module 3 is used as a separate software timer to transmit bits.

Listing 2. Initialization Routine

```

ORG 0000H
LJMP INITIALIZE
ORG 001BH
LJMP RECEIVE_DONE
; Timer 1 overflow -
; simulates "RI" interrupt

ORG 0033H
LJMP RECEIVE
; PCA interrupt

; Initialize stack pointer
INITIALIZE: MOV SP, #5FH
; (specific to test program)

INIT_PCA: MOV CMOD, #00H
; Increment PCA timer
; @ 1/12 Osc Frequency
MOV CCON, #00H
; Clear all status flags
MOV CCAPM0, #NEG_EDGE
; Module 0 in negative-edge
; trigger mode (P1.3)
MOV CCAPM3, #S_W_TIMER
; Module 3 as software timer
; mode

MOV CL, #00H
MOV CH, #00H
MOV IE, #0D8H
; Init all needed interrupts
; EA, EC, ES, ET1
SETB CR
; Turn on PCA Counter

```

270531-5

All flags and registers from Listing 1 should be cleared in the initialization process.

Receive Routine

Two operating modes of the PCA are needed to receive bits. The module must first be able to detect the leading edge of a start bit so it is initially set up to capture a 1-to-0 transition (i.e. Negative-Edge Capture mode). The module is then reconfigured as a software timer to cause an interrupt at the center of each bit to deserialize the incoming data. The flowchart for the receive routine is given in Figure 1.

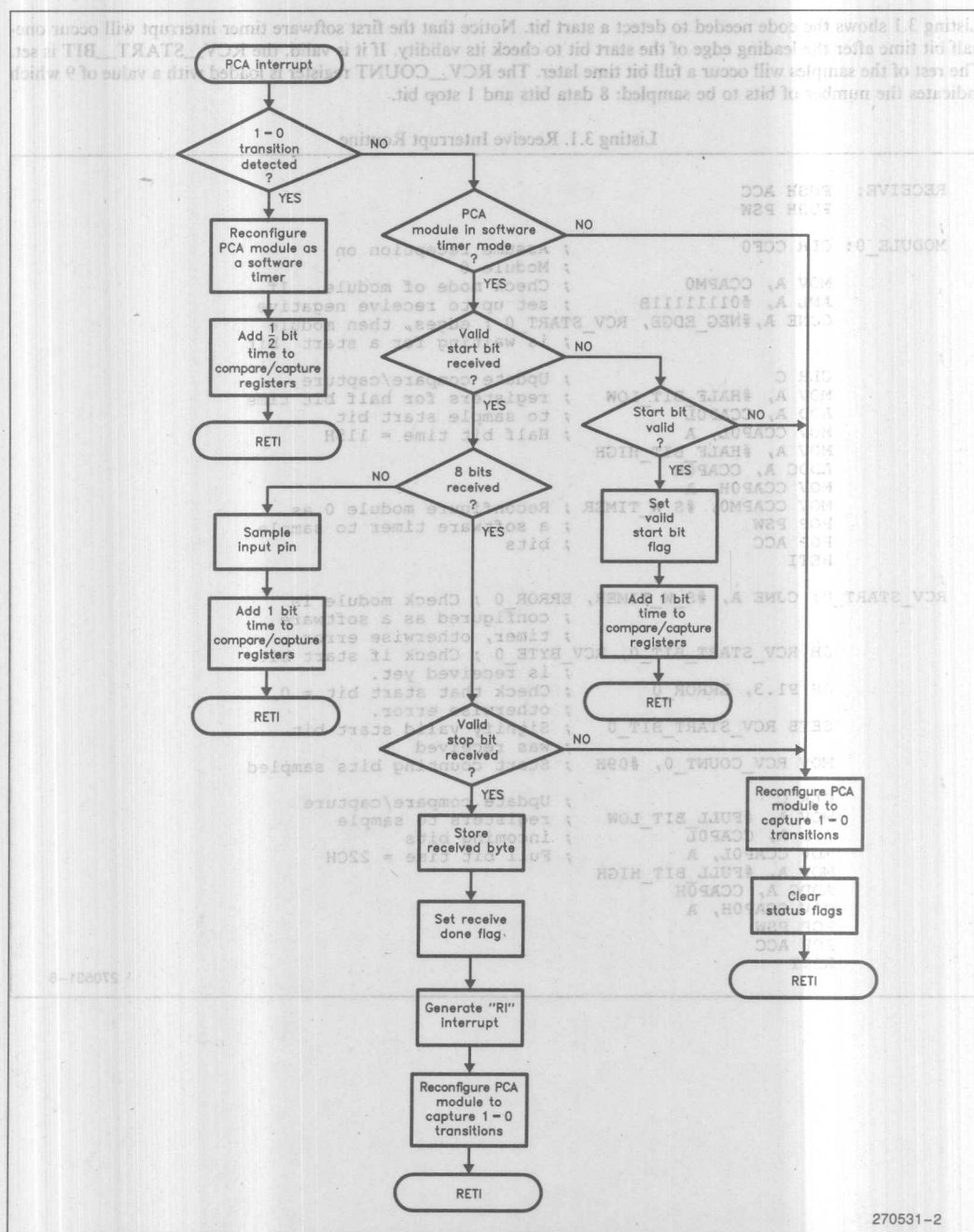


Figure 1. Flowchart for the Receive Routine

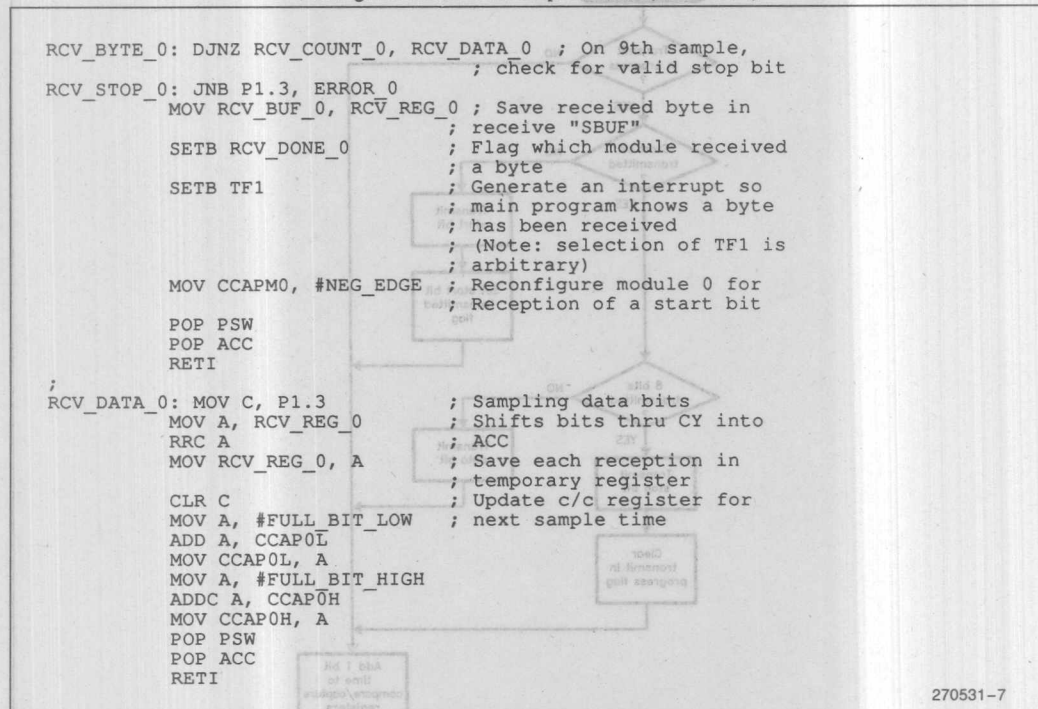
270531-2

Listing 3.1. Receive Interrupt Routine

270531-6

The next 8 timer interrupts will receive the incoming data bits; the RCV_COUNT register keeps track of how many bits have been sampled. As each bit is sampled, it is shifted through the Carry Flag and saved in RCV_REG. The ninth sample checks the validity of the stop bit. If it is valid, the data byte is moved into RCV_BUF. Another interrupt one or more bits can be transmitted through port pins. In the test program three channels were operated. The main routine must have a way to know that a byte has been received. With the on-chip UART, the RI (Receive Interrupt) bit is set whenever a byte has been received. For the software serial port, any unimplemented interrupt vector can be used to generate an interrupt when a byte has been received. This routine uses the Timer 1 Overflow interrupt (its selection is arbitrary). A routine to test this interrupt is included in the listing in the Appendix.

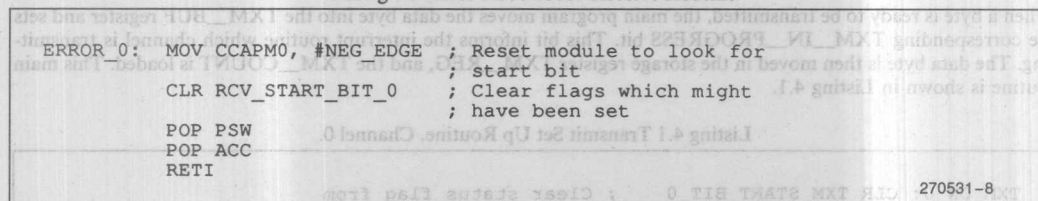
Listing 3.2. Receive Interrupt Routine (Continued)



270531-7

In addition, an error routine (Listing 3.3) is included for invalid start or stop bits to offer some protection against noise. If an error occurs, the module is re-initialized to look for another start bit.

Listing 3.3 Error Routine for Receive Routine



270531-8

Transmit Routine

Another PCA module is configured as a software timer to interrupt the CPU every bit time. With each timer interrupt one or more bits can be transmitted through port pins. In the test program three channels were operated simultaneously, but in the listings below, one channel is shown for simplicity. The selection of port pins is user programmable. The flowchart for the transmit routine is given in Figure 2.

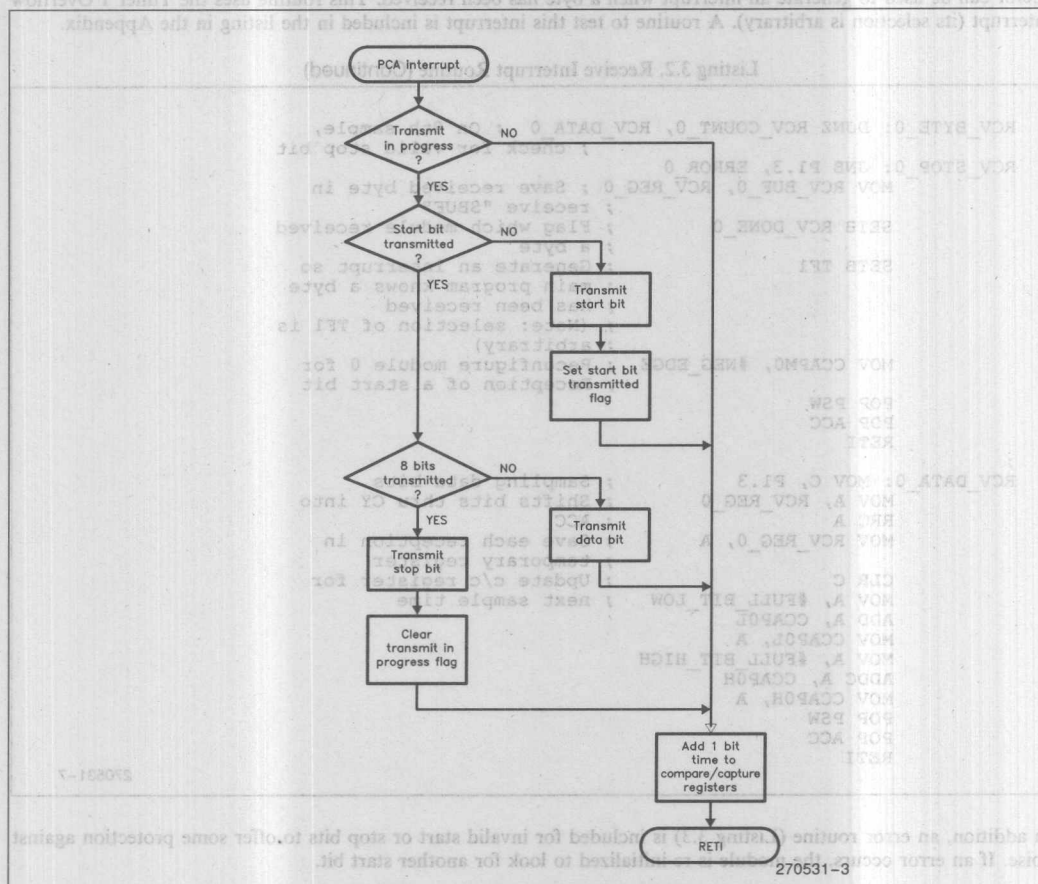


Figure 2. Flowchart for the Transmit Routine

When a byte is ready to be transmitted, the main program moves the data byte into the TXM_BUF register and sets the corresponding TXM_IN_PROGRESS bit. This bit informs the interrupt routine which channel is transmitting. The data byte is then moved in the storage register TXM_REG, and the TXM_COUNT is loaded. This main routine is shown in Listing 4.1.

Listing 4.1 Transmit Set Up Routine. Channel 0.

```

TXM_ON_0: CLR TXM_START_BIT_0    ; Clear status flag from
                                ; previous transmission
                                ; Load "SBUF" with data byte
MOV TXM_BUF_0, DATA_0
MOV TXM_REG_0, TXM_BUF_0
MOV TXM_COUNT_0, #09           ; 8 data bits + 1 stop bit
SETB TXM_IN_PROGRESS_0
  
```

270531-9

Listing 4.2 shows the transmit interrupt routine. The first time through, the start bit is transmitted. As each successive interrupt outputs a bit, the contents of TXM_REG is shifted right one place into the Carry flag, and the TXM_COUNT is decremented. When TXM_COUNT equals zero, the stop bit is transmitted.

Listing 4.2. Transmit Interrupt Routine

```

TRANSMIT: PUSH ACC
          PUSH PSW
          CLR CCF3
          ; Clear s/w timer interrupt
          ; for transmitting bits
          JNB TXM_IN_PROGRESS_0, TRANSMIT_1 ; Check which
          ; channel is transmitting.
          ; "TRANSMIT 1" is listed in
          ; the Appendix
;
TRANSMIT_0: JB TXM_START_BIT_0, TXM_BYTE_0 ; If start bit
          ; has been sent, continue
          ; transmitting bits.
          CLR P3.2 ; Otherwise transmit start
          ; bit
          SETB TXM_START_BIT_0 ; Signify start bit sent
          JMP TXM_EXIT
;
TXM_BYTE_0: DJNZ TXM_COUNT_0, TXM_DATA_0 ; If bit count
          ; equals 1 thru 9, transmit
          ; data bits (8 total)
;
TXM_STOP_0: SETB P3.2 ; When bit count = 0,
          ; transmit stop bit
          CLR TXM_IN_PROGRESS_0 ; Indicate transmission is
          ; finished and ready for
          ; next byte
          JMP TXM_EXIT
;
TXM_DATA_0: MOV A, TXM_REG_0 ; Transmit one bit at a time
          RRC A ; through the carry bit
          MOV P3.2, C
          MOV TXM_REG_0, A ; Save what's not been sent
;
TXM_EXIT: CLR C ; Update compare value with
          MOV A, #FULL_BIT_LOW ; Full bit time = 22CH
          ADD A, CCAP3L
          MOV CCAP3L, A
          MOV A, #FULL_BIT_HIGH
          ADDC A, CCAP3H
          MOV CCAP3H, A
          POP PSW
          POP ACC
          RETI

```

270531-10

Conclusion

The software routines in the Appendix can be altered to vary the baud rate and number of channels to fit a particular application. The number of channels which can be implemented is limited by the CPU time required to service the PCA interrupt. At higher baud rates, fewer channels can be run.

The test program verifies the simultaneous operation of three half-duplex channels at 2400 Baud and the on-chip full-duplex channel at 9600 Baud. Thirty-three percent of the CPU time is required to operate all four channels. The test was run for several hours with no apparent malfunctions.

APPENDIX

Software Interrupts

270531-11

01/01/80 PAGE 1

MCS-51 MACRO ASSEMBLER SWPORT

DOS 3.20 (038-N) MCS-51 MACRO ASSEMBLER, V2.2
OBJECT MODULE PLACED IN SWPORT.OBJ
ASSEMBLER INVOKED BY: C:\AEDIT\ASM51.EXE SWPORT.RCV

```

LOC OBJ      LINE      SOURCE
1            $NOMOD51
2            $NOSYMBOLS
3            $NOLIST
152          ;
153          ;
154          ;
155          ; This program tests the receive routines of a software serial port.
156          ; Three half-duplex channels are implemented in software to run at
157          ; 2400 Baud (16MHz). The on-chip serial port is also running full-duplex
158          ; at 9600 Baud. Thirty-three percent of the CPU time is required to run
159          ; all four ports simultaneously.
160          ;
161          ; To test the receive routines, "dummy" terminals transmit 00 - FF hex
162          ; continually to the PCA. When the first byte is received, it is
163          ; compared with 00. If the comparison is valid, the compare value is
164          ; incremented and the routine waits to receive the next byte. Error
165          ; routines toggle various Port 3 pins if an invalid comparison occurs
166          ; or if an invalid start bit or stop bit is received.
167          ;
0000          ORG 00H
0000 020036    LJMP INITIALIZE
170          ;
001B          ORG 001BH
001B 02025C    LJMP RECEIVE_DONE          ; Timer 1 Overflow - simulates "RI" interrupt
173          ;
0023          ORG 0023H
0023 020282    LJMP SERIAL_PORT          ; Serial port interrupt
176          ;
0033          ORG 0033H
0033 0200DC    LJMP RECEIVE              ; PCA interrupt
178          ;
179          ;
180          ;
181          ;
182          ;
183          ;
184          ;
185          ;
186          ;
187          ;
188          ;
189          ;
190          ;
191          ;
192          ;
193          ;
194          ;
195          ;
196          ;
197          ;
198          ;
199          ;
200          ;
201          ;
202          ;
203          ;
204          ;
205          ;
206          ;
207          ;
208          ;
209          ;
210          ;
211          ;
212          ;
213          ;
214          ;
215          ;
216          ;
217          ;
218          ;
219          ;
220          ;
221          ;
222          ;
223          ;
224          ;
225          ;
226          ;
227          ;
228          ;
229          ;
230          ;
231          ;
232          ;
233          ;
234          ;
235          ;
236          ;
237          ;
238          ;
239          ;
240          ;
241          ;
242          ;
243          ;
244          ;
245          ;
246          ;
247          ;
248          ;
249          ;
250          ;
251          ;
252          ;
253          ;
254          ;
255          ;
256          ;
257          ;
258          ;
259          ;
260          ;
261          ;
262          ;
263          ;
264          ;
265          ;
266          ;
267          ;
268          ;
269          ;
270          ;
271          ;
272          ;
273          ;
274          ;
275          ;
276          ;
277          ;
278          ;
279          ;
280          ;
281          ;
282          ;
283          ;
284          ;
285          ;
286          ;
287          ;
288          ;
289          ;
290          ;
291          ;
292          ;
293          ;
294          ;
295          ;
296          ;
297          ;
298          ;
299          ;
300          ;
301          ;
302          ;
303          ;
304          ;
305          ;
306          ;
307          ;
308          ;
309          ;
310          ;
311          ;
312          ;
313          ;
314          ;
315          ;
316          ;
317          ;
318          ;
319          ;
320          ;
321          ;
322          ;
323          ;
324          ;
325          ;
326          ;
327          ;
328          ;
329          ;
330          ;
331          ;
332          ;
333          ;
334          ;
335          ;
336          ;
337          ;
338          ;
339          ;
340          ;
341          ;
342          ;
343          ;
344          ;
345          ;
346          ;
347          ;
348          ;
349          ;
350          ;
351          ;
352          ;
353          ;
354          ;
355          ;
356          ;
357          ;
358          ;
359          ;
360          ;
361          ;
362          ;
363          ;
364          ;
365          ;
366          ;
367          ;
368          ;
369          ;
370          ;
371          ;
372          ;
373          ;
374          ;
375          ;
376          ;
377          ;
378          ;
379          ;
380          ;
381          ;
382          ;
383          ;
384          ;
385          ;
386          ;
387          ;
388          ;
389          ;
390          ;
391          ;
392          ;
393          ;
394          ;
395          ;
396          ;
397          ;
398          ;
399          ;
400          ;
401          ;
402          ;
403          ;
404          ;
405          ;
406          ;
407          ;
408          ;
409          ;
410          ;
411          ;
412          ;
413          ;
414          ;
415          ;
416          ;
417          ;
418          ;
419          ;
420          ;
421          ;
422          ;
423          ;
424          ;
425          ;
426          ;
427          ;
428          ;
429          ;
430          ;
431          ;
432          ;
433          ;
434          ;
435          ;
436          ;
437          ;
438          ;
439          ;
440          ;
441          ;
442          ;
443          ;
444          ;
445          ;
446          ;
447          ;
448          ;
449          ;
450          ;
451          ;
452          ;
453          ;
454          ;
455          ;
456          ;
457          ;
458          ;
459          ;
460          ;
461          ;
462          ;
463          ;
464          ;
465          ;
466          ;
467          ;
468          ;
469          ;
470          ;
471          ;
472          ;
473          ;
474          ;
475          ;
476          ;
477          ;
478          ;
479          ;
480          ;
481          ;
482          ;
483          ;
484          ;
485          ;
486          ;
487          ;
488          ;
489          ;
490          ;
491          ;
492          ;
493          ;
494          ;
495          ;
496          ;
497          ;
498          ;
499          ;
500          ;
501          ;
502          ;
503          ;
504          ;
505          ;
506          ;
507          ;
508          ;
509          ;
510          ;
511          ;
512          ;
513          ;
514          ;
515          ;
516          ;
517          ;
518          ;
519          ;
520          ;
521          ;
522          ;
523          ;
524          ;
525          ;
526          ;
527          ;
528          ;
529          ;
530          ;
531          ;
532          ;
533          ;
534          ;
535          ;
536          ;
537          ;
538          ;
539          ;
540          ;
541          ;
542          ;
543          ;
544          ;
545          ;
546          ;
547          ;
548          ;
549          ;
550          ;
551          ;
552          ;
553          ;
554          ;
555          ;
556          ;
557          ;
558          ;
559          ;
560          ;
561          ;
562          ;
563          ;
564          ;
565          ;
566          ;
567          ;
568          ;
569          ;
570          ;
571          ;
572          ;
573          ;
574          ;
575          ;
576          ;
577          ;
578          ;
579          ;
580          ;
581          ;
582          ;
583          ;
584          ;
585          ;
586          ;
587          ;
588          ;
589          ;
590          ;
591          ;
592          ;
593          ;
594          ;
595          ;
596          ;
597          ;
598          ;
599          ;
600          ;
601          ;
602          ;
603          ;
604          ;
605          ;
606          ;
607          ;
608          ;
609          ;
610          ;
611          ;
612          ;
613          ;
614          ;
615          ;
616          ;
617          ;
618          ;
619          ;
620          ;
621          ;
622          ;
623          ;
624          ;
625          ;
626          ;
627          ;
628          ;
629          ;
630          ;
631          ;
632          ;
633          ;
634          ;
635          ;
636          ;
637          ;
638          ;
639          ;
640          ;
641          ;
642          ;
643          ;
644          ;
645          ;
646          ;
647          ;
648          ;
649          ;
650          ;
651          ;
652          ;
653          ;
654          ;
655          ;
656          ;
657          ;
658          ;
659          ;
660          ;
661          ;
662          ;
663          ;
664          ;
665          ;
666          ;
667          ;
668          ;
669          ;
670          ;
671          ;
672          ;
673          ;
674          ;
675          ;
676          ;
677          ;
678          ;
679          ;
680          ;
681          ;
682          ;
683          ;
684          ;
685          ;
686          ;
687          ;
688          ;
689          ;
690          ;
691          ;
692          ;
693          ;
694          ;
695          ;
696          ;
697          ;
698          ;
699          ;
700          ;
701          ;
702          ;
703          ;
704          ;
705          ;
706          ;
707          ;
708          ;
709          ;
710          ;
711          ;
712          ;
713          ;
714          ;
715          ;
716          ;
717          ;
718          ;
719          ;
720          ;
721          ;
722          ;
723          ;
724          ;
725          ;
726          ;
727          ;
728          ;
729          ;
730          ;
731          ;
732          ;
733          ;
734          ;
735          ;
736          ;
737          ;
738          ;
739          ;
740          ;
741          ;
742          ;
743          ;
744          ;
745          ;
746          ;
747          ;
748          ;
749          ;
750          ;
751          ;
752          ;
753          ;
754          ;
755          ;
756          ;
757          ;
758          ;
759          ;
760          ;
761          ;
762          ;
763          ;
764          ;
765          ;
766          ;
767          ;
768          ;
769          ;
770          ;
771          ;
772          ;
773          ;
774          ;
775          ;
776          ;
777          ;
778          ;
779          ;
780          ;
781          ;
782          ;
783          ;
784          ;
785          ;
786          ;
787          ;
788          ;
789          ;
790          ;
791          ;
792          ;
793          ;
794          ;
795          ;
796          ;
797          ;
798          ;
799          ;
800          ;
801          ;
802          ;
803          ;
804          ;
805          ;
806          ;
807          ;
808          ;
809          ;
810          ;
811          ;
812          ;
813          ;
814          ;
815          ;
816          ;
817          ;
818          ;
819          ;
820          ;
821          ;
822          ;
823          ;
824          ;
825          ;
826          ;
827          ;
828          ;
829          ;
830          ;
831          ;
832          ;
833          ;
834          ;
835          ;
836          ;
837          ;
838          ;
839          ;
840          ;
841          ;
842          ;
843          ;
844          ;
845          ;
846          ;
847          ;
848          ;
849          ;
850          ;
851          ;
852          ;
853          ;
854          ;
855          ;
856          ;
857          ;
858          ;
859          ;
860          ;
861          ;
862          ;
863          ;
864          ;
865          ;
866          ;
867          ;
868          ;
869          ;
870          ;
871          ;
872          ;
873          ;
874          ;
875          ;
876          ;
877          ;
878          ;
879          ;
880          ;
881          ;
882          ;
883          ;
884          ;
885          ;
886          ;
887          ;
888          ;
889          ;
890          ;
891          ;
892          ;
893          ;
894          ;
895          ;
896          ;
897          ;
898          ;
899          ;
900          ;
901          ;
902          ;
903          ;
904          ;
905          ;
906          ;
907          ;
908          ;
909          ;
910          ;
911          ;
912          ;
913          ;
914          ;
915          ;
916          ;
917          ;
918          ;
919          ;
920          ;
921          ;
922          ;
923          ;
924          ;
925          ;
926          ;
927          ;
928          ;
929          ;
930          ;
931          ;
932          ;
933          ;
934          ;
935          ;
936          ;
937          ;
938          ;
939          ;
940          ;
941          ;
942          ;
943          ;
944          ;
945          ;
946          ;
947          ;
948          ;
949          ;
950          ;
951          ;
952          ;
953          ;
954          ;
955          ;
956          ;
957          ;
958          ;
959          ;
960          ;
961          ;
962          ;
963          ;
964          ;
965          ;
966          ;
967          ;
968          ;
969          ;
970          ;
971          ;
972          ;
973          ;
974          ;
975          ;
976          ;
977          ;
978          ;
979          ;
980          ;
981          ;
982          ;
983          ;
984          ;
985          ;
986          ;
987          ;
988          ;
989          ;
990          ;
991          ;
992          ;
993          ;
994          ;
995          ;
996          ;
997          ;
998          ;
999          ;
1000         ;

```

VARIABLES USED BY THE SOFTWARE SERIAL PORT

RECEIVE ROUTINE:

```

RCV_START_BIT_0    BIT    20H.0    ; Indicates start bit has been
RCV_START_BIT_1    BIT    21H.0    ; received
RCV_START_BIT_2    BIT    22H.0
RCV_DONE_0         BIT    20H.1    ; Indicates data byte has been
RCV_DONE_1         BIT    21H.1    ; received
RCV_DONE_2         BIT    22H.1
RCV_ON_0           BIT    20H.2    ; Used in main test program to check
RCV_ON_1           BIT    21H.2    ; for a received byte
RCV_ON_2           BIT    22H.2

```


LOC	OBJ	LINE	SOURCE	DATA	30H	
0030		199	RCV_BUF_0	DATA	30H	; Software receive "SBUF"
0040		200	RCV_BUF_1	DATA	40H	
0050		201	RCV_BUF_2	DATA	50H	
0031		202	;			
0041		203	RCV_REG_0	DATA	31H	; Temporary register for
0051		204	RCV_REG_1	DATA	41H	; receiving bits
0032		205	RCV_REG_2	DATA	51H	
0042		206	;			
0052		207	RCV_COUNT_0	DATA	32H	; Counter for receiving bits
0043		208	RCV_COUNT_1	DATA	42H	
0053		209	RCV_COUNT_2	DATA	52H	
0033		210	;			
0043		211	COUNT_0	DATA	33H	; Used in test program to check
0053		212	COUNT_1	DATA	43H	; bytes being received
0011		213	COUNT_2	DATA	53H	
0049		214	;			
0015		215	NEG_EDGE	EQU	11H	; Two modes of operation for the
0001		216	S_W_TIMER	EQU	49H	; Compare/Capture modules
002C		217	;			
0002		218	HALF_BIT_LOW	EQU	15H	; Half bit time = 115H
		219	HALF_BIT_HIGH	EQU	01H	
		220	FULL_BIT_LOW	EQU	2CH	; Full bit time = 22CH
		221	FULL_BIT_HIGH	EQU	02H	
		222	;			; 2400 Baud @ 16MHz
		223	;			
		224	;			
		225	;			
		226	;			
		227	;			
		228	;			
		229	;			
		230	INITIALIZE:	MOV SP, #5FH		; Initialize stack pointer
0036 75815F		231				; (specific to the test program)
0039 75D900		232	INIT_PCA:	MOV CMOD, #00H		; Increment PCA clock @ 1/12 Osc Freq
003C 75D800		233		MOV CCON, #00H		; Clear all status flags
003F 75DA11		234		MOV CCAPM0, #NEG_EDGE		; Module 0 in Neg-edge capture mode (Pl.3)
0042 75DB11		235		MOV CCAPM1, #NEG_EDGE		; Module 1 " (Pl.4)
0045 75DC11		236		MOV CCAPM2, #NEG_EDGE		; Module 2 " (Pl.5)
		237	;			
0048 75E900		238		MOV CL, #00H		
004B 75F900		239		MOV CH, #00H		
004E 75A8D8		240		MOV IE, #0D8H		; Initialize needed interrupt: EA, EC, ES, ETI
0051 D2DE		241		SETB CR		; Turn on PCA counter
		242	;			
0053 759850		243	INIT_SP:	MOV SCON, #50H		; Serial port in mode 1 (8-Bit UART)
0056 75CBFF		244		MOV RCAP2H, #0FFH		; Reload values for 9600 Baud @ 16 MHz
0059 75CACC		245		MOV RCAP2L, #0CCH		
005C 75C834		246		MOV T2CON, #34H		; Timer 2 as a baud-rate generator,
		247				; turn on timer 2
		248	;			
005F C200		249	INIT_FLAGS:	CLR RCV_START_BIT_0		
0061 C208		250		CLR RCV_START_BIT_1		
0063 C210		251		CLR RCV_START_BIT_2		
		252	;			
0065 C201		253		CLR RCV_DONE_0		

270531-12

intel

AB-41

16M

2-232

MCS-51 MACRO ASSEMBLER SWPORT

01/01/80 PAGE 4

```

LOC OBJ          LINE    SOURCE
00C1 C212        309      CLR RCV ON 2
00C3 0553        310      INC COUNT 2
00C5 80DA        311      JMP CHECK_0
                        312
00C7 C2B2        313      ERROR0: CLR P3.2          ; Error in comparison on module 0
00C9 75DA00      314      MOV CCAPM0, #00H      ; Discontinue receiving bytes
00CC 80DF        315      JMP CHECK_1
                        316
00CE C2B3        317      ERROR1: CLR P3.3          ; Error in comparison on module 1
00D0 75DB00      318      MOV CCAPM1, #00H
00D3 80E4        319      JMP CHECK_2
                        320
00D5 C2B4        321      ERROR2: CLR P3.4          ; Error in comparison on module 2
00D7 75DC00      322      MOV CCAPM2, #00H
00DA 80C5        323      JMP CHECK_0
                        324
                        325
                        326
                        327
                        328
                        329
00DC C0E0        330      RECEIVE: PUSH ACC
00DE C0D0        331      PUSH PSW
                        332
00E0 20D811      333      JB CCF0, MODULE_0      ; Check which module caused
00E3 20D908      334      JB CCF1, JUMP_1      ; PCA interrupt and jump to
00E6 20DA08      335      JB CCF2, JUMP_2      ; appropriate routine
00E9 D0D0        336      POP PSW
00EB D0E0        337      POP ACC
00ED 32          338      RETI
                        339
00EE 02016C      340      JUMP_1: LJMP MODULE_1
00F1 0201E4      341      JUMP_2: LJMP MODULE_2
                        342
00F3 20B358      343      ; PCA B206 01
                        344
00F5 D2A513      345      ; PCA B206 01
                        346
00F7 75        347      ; PCA B206 01
                        348
00F4 C2D8        349      MODULE_0: CLR CCF0
00F6 E5DA        350      MOV A, CCAPM0          ; Reception on module 0
00F8 547F        351      ANL A, #01111111B      ; Check mode of module. If set up to
00FA B41115      352      CJNE A, #NEG_EDGE, RCV_START_0 ; receive negative edges, then module
                        353      ; is waiting for a start bit
00FD C3          354      CLR C
00FE 7415        355      MOV A, #HALF_BIT_LOW    ; Update Compare/Capture registers for
0100 25EA        356      ADD A, CCAP0L          ; half a bit time
0102 F5EA        357      MOV CCAP0L, A          ; to sample start bit
0104 7401        358      MOV A, #HALF_BIT_HIGH   ; Half bit time = 115H
0106 35FA        359      ADDC A, CCAP0H
0108 F5FA        360      MOV CCAP0H, A
010A 75DA49      361      MOV CCAPM0, #S_W_TIMER ; Reconfigure module 0 as
010D D0D0        362      POP PSW              ; a software timer to sample bits
010F D0E0        363      POP ACC

```

270531-14

```

LOC  OBJ      LINE  SOURCE
0111  32        364      RETI
0112  B4494B    365      ; RCV_START_0: CJNE A, $S_W_TIMER, ERROR_0 ; Check module is configured
                                ; as a software timer, otherwise error.
0115  20001A    366      JB RCV_START_BIT_0, RCV_BYTE_0 ; Check if start bit
                                ; has been received yet
0118  209345    367      JB P1.3, ERROR_0 ; Check that start bit = 0,
                                ; otherwise error.
011B  D200      370      SETB RCV_START_BIT_0 ; Signify valid start bit
                                ; was received
011D  753209    371      MOV RCV_COUNT_0, #09H ; Start counting bits sampled
                                ;
0120  C3        372      CLR C ; Update C/C registers to sample
0121  742C      373      MOV A, #FULL_BIT_LOW ; incoming bits
0123  25EA      374      ADD A, CCAP0L ; Full bit time = 22CH
0125  F5EA      375      MOV CCAP0L, A
0127  7402      376      MOV A, #FULL_BIT_HIGH
0129  35FA      377      ADDC A, CCAP0H
012B  F5FA      378      MOV CCAP0H, A
012D  D0D0      379      POP PSM
012F  D0E0      380      POP ACC
0131  32        381      RETI
0132  D53212    382      ; RCV_BYTE_0: DJNZ RCV_COUNT_0, RCV_DATA_0 ; On 9th sample, check for
                                ; valid stop bit
0135  309328    383      RCV_STOP_0: JNB P1.3, ERROR_0 ; Save received byte in receive "SBUF"
0138  853130    384      MOV RCV_BUF_0, RCV_REG_0 ; Flag which module received a byte
013B  D201      385      SETB RCV_DONE_0 ; Generate an interrupt so main program
013D  D28F      386      SETB TFI ; knows a byte has been received
                                ; (NOTE: selection of TFI is arbitrary)
013F  75DA11    387      MOV CCAPM0, #NEG_EDGE ; Reconfigure module 0 for next
0142  D0D0      388      POP PSM ; reception of a start bit
0144  D0E0      389      POP ACC
0146  32        390      RETI
0147  A293      391      ; RCV_DATA_0: MOV C, P1.3 ; Sampling data bits
0149  E531      392      MOV A, RCV_REG_0 ; Shift bits through CY into ACC
014B  13        393      RRC A
014C  F531      394      MOV RCV_REG_0, A ; Save each reception in temporary
                                ; register
014E  C3        395      CLR C ; Update C/C register for next
014F  742C      396      MOV A, #FULL_BIT_LOW ; sample time
0151  25EA      397      ADD A, CCAP0L
0153  F5EA      398      MOV CCAP0L, A
0155  7402      399      MOV A, #FULL_BIT_HIGH
0157  35FA      400      ADDC A, CCAP0H
0159  F5FA      401      MOV CCAP0H, A
015B  D0D0      402      POP PSM
015D  D0E0      403      POP ACC
015F  32        404      RETI
0160  C2B5      405      ; ERROR_0: CLR P3.5 ; Error routine for invalid start or
                                ; stop bit or invalid mode comparison
0162  32B5      406      MOV P3_5, #0
0164  32B5      407      MOV P3_5, #0
0166  32B5      408      MOV P3_5, #0
0168  32B5      409      MOV P3_5, #0
016A  32B5      410      MOV P3_5, #0
016C  32B5      411      MOV P3_5, #0
016E  32B5      412      MOV P3_5, #0
016F  32B5      413      MOV P3_5, #0
0170  32B5      414      MOV P3_5, #0
0171  32B5      415      MOV P3_5, #0
0172  32B5      416      MOV P3_5, #0
0173  32B5      417      MOV P3_5, #0

```

270531-15

intel

AB-41

1011

LOC	OBJ	LINE	SOURCE
0162	75DA11	418	
0165	C200	419	MOV CCAPMO, #NEG_EDGE ; Port pin used for debug only
0167	D0D0	420	CLR RCV_START_BIT_0 ; Reset module to look for start bit
0169	D0E0	421	POP PSM ; Clear flags which might have been set
016B	32	422	POP ACC
		423	RETI
		424	
		425	;
		426	;
		427	;
		428	CHANNEL_1
		429	-----
016C	C2D9	429	MODULE_1: CLR CCF1 ; Similar to module 0
016E	E5DB	430	MOV A, CCAPM1
0170	547F	431	ANL A, #01111111B
0172	B41115	432	CJNE A, #NEG_EDGE, RCV_START_1
		433	;
0175	C3	434	CLR C
0176	7415	435	MOV A, #HALF_BIT_LOW
0178	25EB	436	ADD A, CCAPIL
017A	F5EB	437	MOV CCAPIL, A
017C	7401	438	MOV A, #HALF_BIT_HIGH
017E	35FB	439	ADDC A, CCAPIL
0180	F5FB	440	MOV CCAPIL, A
0182	75DB49	441	MOV CCAPM1, #S_W_TIMER
0185	D0D0	442	POP PSM
0187	D0E0	443	POP ACC
0189	32	444	RETI
		445	;
018A	B4494B	446	RCV_START_1: CJNE A, #S_W_TIMER, ERROR_1
018D	20081A	447	JB RCV_START_BIT_1, RCV_BYTE_1
0190	209445	448	JB P1.4, ERROR_1
		449	;
0193	D208	450	SETB RCV_START_BIT_1
0195	754209	451	MOV RCV_COUNT_1, #09H
		452	;
		453	CLR C
0198	C3	454	MOV A, #FULL_BIT_LOW
0199	742C	455	ADD A, CCAPIL
019B	25EB	456	MOV CCAPIL, A
019D	F5EB	457	MOV A, #FULL_BIT_HIGH
019F	7402	458	ADDC A, CCAPIL
01A1	35FB	459	MOV CCAPIL, A
01A3	F5FB	460	POP PSM
01A5	D0D0	461	POP ACC
01A7	D0E0	462	RETI
01A9	32	463	
01AA	D54212	464	RCV_BYTE_1: DJNZ RCV_COUNT_1, RCV_DATA_1
		465	;
01AD	309428	466	RCV_STOP_1: JNB P1.4, ERROR_1
01B0	854140	467	MOV RCV_BUF_1, RCV_REG_1
01B3	D209	468	SETB RCV_DONE_1
01B5	D28F	469	SETB RETI
01B7	75DB11	470	MOV CCAPM1, #NEG_EDGE
01BA	D0D0	471	POP PSM
01BC	D0E0	472	POP ACC

270531-16

intel

AB-41

16871

MCS-51 MACRO ASSEMBLER SWPORT

01/01/80 PAGE 7

LOC	OBJ	LINE	SOURCE
01BE	32	473	RETI
01BF	A294	474	
01C1	E341	475	RCV_DATA_1:
01C3	13	476	MOV C, P1.4
01C4	F541	477	MOV A, RCV_REG_1
		478	RRC A
		479	MOV RCV_REG_1, A
01C6	C3	480	CLR C
01C7	742C	481	MOV A, #FULL BIT_LOW
01C9	25EB	482	ADD A, CCAP1L
01CB	F5EB	483	MOV CCAP1L, A
01CD	7402	484	MOV A, #FULL BIT_HIGH
01CF	35FB	485	ADDC A, CCAP1H
01D1	F5FB	486	MOV CCAP1H, A
01D3	D0D0	487	POP PSW
01D5	D0E0	488	POP ACC
01D7	32	489	RETI
		490	
01D8	C2B6	491	ERROR_1:
01DA	75DB11	492	CLR P3.6
01DD	C208	493	MOV CCAPM1, #NEG_EDGE
01DF	D0D0	494	CLR RCV_START_BIT_1
01E1	D0E0	495	POP PSW
01E3	32	496	POP ACC
		497	RETI
		498	
		499	
		500	CHANNEL 2
		501	
		502	
		503	
01E4	C2DA	504	MODULE_2:
01E6	E5DC	505	CLR CCF2
01E8	547F	506	MOV A, CCAPM2
01EA	B41115	507	ANL A, #01111111B
		508	CJNE A, #NEG_EDGE, RCV_START_2
		509	
01ED	C3	510	CLR C
01EE	7415	511	MOV A, #HALF_BIT_LOW
01F0	25EC	512	ADD A, CCAP2L
01F2	F5EC	513	MOV CCAP2L, A
01F4	7401	514	MOV A, #HALF_BIT_HIGH
01F6	35FC	515	ADDC A, CCAP2H
01F8	F5FC	516	MOV CCAP2H, A
01FA	75DC49	517	MOV CCAPM2, #S_W_TIMER
01FD	D0D0	518	POP PSW
01FE	D0E0	519	POP ACC
0201	32	520	RETI
		521	
0202	B4494B	522	RCV_START_2:
0205	20101A	523	CJNE A, #S_W_TIMER, ERROR_2
0208	209545	524	JB RCV_START_BIT_2, RCV_BYTE_2
		525	JB P1.5, ERROR_2
		526	
020B	D210	527	SETB RCV_START_BIT_2
020D	755209		MOV RCV_COUNT_2, #09H

510E31-10

270531-17

intel

AB-41

10711

LOC	OBJ	LINE	SOURCE
0210	C3	528	CLR C
0211	742C	529	MOV A, #FULL_BIT_LOW
0213	25EC	530	ADD A, CCAP2L
0215	F5EC	531	MOV CCAP2L, A
0217	7402	532	MOV A, #FULL_BIT_HIGH
0219	35FC	533	ADDC A, CCAP2H
021B	F5FC	534	MOV CCAP2H, A
021D	D0D0	535	POP PSW
021F	D0E0	536	POP ACC
0221	32	537	RETI
0222	D55212	538	
		539	RCV_BYTE_2: DJNZ RCV_COUNT_2, RCV_DATA_2
		540	
0225	309528	541	RCV_STOP_2: JNB P1.5, ERROR_2
0228	855150	542	MOV RCV_BUF_2, RCV_REG_2
022B	D211	543	SETB RCV_DONE_2
022D	D28F	544	SETB TF1
022F	75DC11	545	MOV CCAPM2, #NEG_EDGE
0232	D0D0	546	POP PSW
0234	D0E0	547	POP ACC
0236	32	548	RETI
		549	
0237	A295	550	RCV_DATA_2: MOV C, P1.5
0239	E551	551	MOV A, RCV_REG_2
023B	13	552	RRC A
023C	F551	553	MOV RCV_REG_2, A
023E	C3	554	CLR C
023F	742C	555	MOV A, #FULL_BIT_LOW
0241	25EC	556	ADD A, CCAP2L
0243	F5EC	557	MOV CCAP2L, A
0245	7402	558	MOV A, #FULL_BIT_HIGH
0247	35FC	559	ADDC A, CCAP2H
0249	F5FC	560	MOV CCAP2H, A
024B	D0D0	561	POP PSW
024D	D0E0	562	POP ACC
024F	32	563	RETI
		564	
0250	C2B7	565	ERROR_2: CLR P3.7
0252	75DC11	566	MOV CCAPM2, #NEG_EDGE
0255	C210	567	CLR RCV_START_BIT_2
0257	D0D0	568	POP PSW
0259	D0E0	569	POP ACC
025B	32	570	RETI
		571	
		572	
		573	
		574	
		575	; This routine simulates the "RI" interrupt. When a byte is received on one
		576	; of the channels, this interrupt is generated. Bits are set so the main
		577	; routine knows which channel received a byte.
		578	
		579	
025C	C0E0	580	RECEIVE_DONE: PUSH ACC
025E	C0D0	581	PUSH PSW
0260	C28F	582	CLR TF1

X10021-18

270531-18

LOC	OBJ	LINE	SOURCE
0262	300106	583	JNB RCV DONE 0, RCV_1 ; Check which module received a byte
0265	C201	584	CLR RCV_DONE 0 ; Clear flags needed for next reception
0267	C200	585	CLR RCV_START BIT 0
0269	D202	586	SETB RCV_ON 0 ; Tell main routine which channel received
		587	; a byte
026B	300906	588	RCV_1: JNB RCV DONE 1, RCV_2
026E	C209	589	CLR RCV_DONE 1
0270	C208	590	CLR RCV_START BIT 1
0272	D20A	591	SETB RCV_ON 1
		592	
0274	301106	593	RCV_2: JNB RCV DONE 2, RETURN
0277	C211	594	CLR RCV_DONE 2
0279	C210	595	CLR RCV_START BIT 2
027B	D212	596	SETB RCV_ON 2
		597	
027D	D0D0	598	RETURN: POP PSW
027F	D0E0	599	POP ACC
0281	32	600	RETI
		601	
		602	
		603	
		604	
		605	SERIAL PORT INTERRUPT
		606	*****
		607	
		608	; When a byte is received on the full-duplex serial port, it is then
		609	; transmitted back to a "dummy" terminal. This terminal checks that the
		610	; byte it transmitted to the PCA is the same value it receives back.
		611	
		612	
0282	C0E0	613	SERIAL_PORT: PUSH ACC
0284	C0D0	614	PUSH PSW
0286	30980B	615	JNB RI, TXM ; Check whether RI or TI
0289	E599	616	MOV A, SBUF ; caused the interrupt
028B	C298	617	CLR RI
028D	F599	618	MOV SBUF, A
028F	D0D0	619	POP PSW
0291	D0E0	620	POP ACC
0293	32	621	RETI
		622	
0294	C299	623	TXM: CLR TI
0296	D0D0	624	POP PSW
0298	D0E0	625	POP ACC
029A	32	626	RETI
		627	
		628	END

REGISTER BANK(S) USED: 0

ASSEMBLY COMPLETE, NO ERRORS FOUND

270531-19

intel

AB-41

1971

MCS-51 MACRO ASSEMBLER SWPORT

01/01/80 PAGE 1 510531-51

DOS 3.20 (038-N) MCS-51 MACRO ASSEMBLER, V2.2
 OBJECT MODULE PLACED IN SWPORT.OBJ
 ASSEMBLER INVOKED BY: C:\AEDIT\ASM51.EXE SWPORT.TR

LOC	OBJ	LINE	SOURCE	
0017	124000	1	\$NOMOD51	
0018	124000	2	\$NOSYMBOLS	
0019	124000	3	\$NOLIST	
0017	124000	152		
0017	124000	153		
0017	124000	154		
0017	124000	155	; This program tests the transmit routines for the software serial port.	
0017	124000	156	; To initialize the first transmission, the compare values are loaded before	
0017	124000	157	; the PCA timer is started. Successive interrupts are generated every bit	
0017	124000	158	; time by the software timer.	
0017	124000	159		
0017	124000	160	; For test purposes, the data transmitted increments from 00 to FF hex.	
0017	124000	161	; "Dummy" terminals receive these bytes and display the bytes as they	
0017	124000	162	; are incremented.	
0017	124000	163		
0000	020036	164	ORG 00H	
0000	020036	165	LJMP INIT_TXM	
0023	02014B	166		
0023	02014B	167	ORG 0023H	
0023	02014B	168	LJMP SERIAL_PORT	; Serial port interrupt
0033	0200D0	169		
0033	0200D0	170	ORG 0033H	
0033	0200D0	171	LJMP TRANSMIT	; PCA software timer interrupt
0033	0200D0	172		
0033	0200D0	173		
0033	0200D0	174		
0033	0200D0	175		
0033	0200D0	176		
0003	0200D0	177	TXM_START_BIT_0	20H.3 ; Indicates start bit has been
000B	0200D0	178	TXM_START_BIT_1	21H.3 ; transmitted
0013	0200D0	179	TXM_START_BIT_2	22H.3
0004	0200D0	180		
000C	0200D0	181	TXM_IN_PROGRESS_0	20H.4 ; Indicates transmit is in progress
0014	0200D0	182	TXM_IN_PROGRESS_1	21H.4
0014	0200D0	183	TXM_IN_PROGRESS_2	22H.4
0034	0200D0	184		
0044	0200D0	185	TXM_BUF_0	34H ; Software transmit "SBUF"
0054	0200D0	186	TXM_BUF_1	44H
0054	0200D0	187	TXM_BUF_2	54H
0035	0200D0	188		
0045	0200D0	189	TXM_REG_0	35H ; Temporary register for
0055	0200D0	190	TXM_REG_1	45H ; transmitting bits
0055	0200D0	191	TXM_REG_2	55H
0036	0200D0	192		
0046	0200D0	193	TXM_COUNT_0	36H ; Counter for transmitting bits
0056	0200D0	194	TXM_COUNT_1	46H
0056	0200D0	195	TXM_COUNT_2	56H
0037	0200D0	196		
0047	0200D0	197	DATA_0	37H ; Register used for the test
0047	0200D0	198	DATA_1	47H ; program

270531-20

MCS-51 MACRO ASSEMBLER SWPORT 01/01/80 PAGE 2 270531-50

LOC	OBJ	LINE	SOURCE	DATA	
0057		199	DATA_2	DATA	57H
0049		200	S_W_TIMER	EQU	49H
002C		202	FULL_BIT_LOW	EQU	2CH
0002		203	FULL_BIT_HIGH	EQU	02H
		204			
		205			
		206			
		207	INITIALIZATION		
		208	=====		
		209			
0036	75815F	210	INIT_TXM: MOV SP, #5FH		
0039	75D900	211	MOV CMOD, #00H		
003C	75D800	212	MOV CCON, #00H		
003F	75F900	213	MOV CH, #00H		
0042	75E900	214	MOV CL, #00H		
0045	75DD49	215	MOV CCAPM3, S_W_TIMER		
0048	75A8D8	216	MOV IE, #0D8H		
004B	759850	217	MOV SCON, #50H		
004E	75CBFF	218	MOV RCAP2H, #0FFH		
0051	75CACC	219	MOV RCAP2L, #0CCH		
0054	75C834	220	MOV T2CON, #34H		
		221	INIT_FLAGS: CLR TXM_START_BIT_0		
		222	CLR TXM_START_BIT_1		
		223	CLR TXM_START_BIT_2		
		224	CLR TXM_IN_PROGRESS_0		
		225	CLR TXM_IN_PROGRESS_1		
		226	CLR TXM_IN_PROGRESS_2		
		227	MOV TXM_BUF_0, #00H		
		228	MOV TXM_BUF_1, #00H		
		229	MOV TXM_BUF_2, #00H		
		230	MOV TXM_REG_0, #00H		
		231	MOV TXM_REG_1, #00H		
		232	MOV TXM_REG_2, #00H		
		233	MOV TXM_COUNT_0, #00H		
		234	MOV TXM_COUNT_1, #00H		
		235	MOV TXM_COUNT_2, #00H		
		236			
		237			
		238			
		239			
		240			
		241			
		242			
		243			
		244			
		245			
		246			
		247			
		248			
		249			
		250			
		251			
		252			
		253			

; (Compatible with receive routines)
 ; Increment PCA timer @ 1/12 osc. freq.
 ; Clear all status flags
 ; Module 3 configured as software timer
 ; Initialize all needed interrupts
 ; Serial port in mode 1 (8-bit UART)
 ; Reload values for 9600 Baud @ 16 MHz
 ; Timer 2 as a baud-rate generator,
 ; turn Timer 2 on
 ; Cause the first software timer to
 ; interrupt one bit time after
 ; PCA timer is started

270531-21

MCS-51 MACRO ASSEMBLER SWPORT

01/01/80 PAGE 3 270531-22

```

LOC OBJ LINE SOURCE
254 ;
255 ;
256 ;
257 ;
258 ;
259 ;
008F 02009D FIRST_TXM: JMP TXM_ON_0
0092 300408 JNB TXM_IN_PROGRESS_0, TXM_ON_0 ; Determine if ready to send
0095 300C16 JNB TXM_IN_PROGRESS_1, TXM_ON_1 ; next byte. (i.e. transmit
0098 301424 JNB TXM_IN_PROGRESS_2, TXM_ON_2 ; "not" in progress)
009B 80F5 JMP MAIN_TXM ; Waiting for "TI" flag
264 ;
265 ;
266 ;
009D C203 TXM_ON_0: CLR TXM_START_BIT_0 ; Clear flag from previous
009F 0537 INC DATA_0 ; transmission
00A1 853734 MOV TXM_BUF_0, DATA_0 ; Load "SBUF" with data byte
00A4 853435 MOV TXM_REG_0, TXM_BUF_0
00A7 753609 MOV TXM_COUNT_0, #09H ; 8 data bits + 1 stop bit
00AA D204 SETB TXM_IN_PROGRESS_0
00AC 80E4 JMP MAIN_TXM
273 ;
274 ;
00AE C20B TXM_ON_1: CLR TXM_START_BIT_1
00B0 0547 INC DATA_1
00B2 854744 MOV TXM_BUF_1, DATA_1
00B5 854445 MOV TXM_REG_1, TXM_BUF_1
00B8 754609 MOV TXM_COUNT_1, #09H
00BB D20C SETB TXM_IN_PROGRESS_1
00BD 80D3 JMP MAIN_TXM
281 ;
282 ;
00BF C213 TXM_ON_2: CLR TXM_START_BIT_2
00C1 0557 INC DATA_2
00C3 855754 MOV TXM_BUF_2, DATA_2
00C6 855455 MOV TXM_REG_2, TXM_BUF_2
00C9 755609 MOV TXM_COUNT_2, #09H
00CC D214 SETB TXM_IN_PROGRESS_2
00CE 80C2 JMP MAIN_TXM
289 ;
290 ;
291 ;
292 ;
293 ;
00D0 C0E0 TRANSMIT: PUSH ACC
00D2 C0D0 PUSH PSW
00D4 C2DB CLR CCF3 ; Clear s/w timer interrupt
00D6 30041E JNB TXM_IN_PROGRESS_0, TRANSMIT_1 ; Check which channel is
298 ; transmitting
299 ;
300 ;
301 ;
302 ;
303 ;
00D9 200307 TRANSMIT_0: JB TXM_START_BIT_0, TXM_BYTE_0 ; If start bit has been sent,
304 ; continue transmitting data bits,
305 ; otherwise transmit start bit
00DC C2B2 CLR P3.2 ; COM2L 0
00DE D203 SETB TXM_START_BIT_0 ; Signify start bit sent
00E0 0200F7 JMP TRANSMIT_1 ; Check next transmit pin
308 ;

```

MCS-51 MACRO ASSEMBLER SWPORT

01/01/80 PAGE 3 270531-22

MCS-51 MACRO ASSEMBLER SWPORT

01/01/80 PAGE 4 510031-55

```

LOC OBJ LINE SOURCE
00E3 D53607 309 TXM_BYTE_0: DJNZ TXM_COUNT_0, TXM_DATA_0
00E6 D2B2 310 TXM_STOP_0: SETB P3.2
00E8 C204 311 CLR TXM_IN_PROGRESS_0
00EA 0200F7 312 JMP TRANSMIT_1
00ED E535 313 TXM_DATA_0: MOV A, TXM_REG_0
00EF 13 314 RRC A
00F0 92B2 315 MOV P3.2, C
00F2 F535 316 MOV TXM_REG_0, A
00F4 0200F7 317 JMP TRANSMIT_1
; CHANNEL 1
; TXM_BYTE_1: DJNZ TXM_COUNT_1, TXM_DATA_1
; TXM_STOP_1: SETB P3.3
; TXM_DATA_1: MOV A, TXM_REG_1
; TRANSMIT_2: JNB TXM_IN_PROGRESS_2, TXM_EXIT
; TXM_BYTE_2: DJNZ TXM_COUNT_2, TXM_DATA_2
; TXM_STOP_2: SETB P3.4
; TXM_DATA_2: MOV A, TXM_REG_2
; TXM_EXIT:

```

MCS-51 MACRO ASSEMBLER SWPORT

01/01/80 PAGE 4 270531-23

LOC	OBJ	LINE	SOURCE
0139	C3	364	TXM_EXIT: CLR C ; Update compare value with
013A	742C	365	MOV A, #FULL BIT LOW ; full bit time = 22CH
013C	25ED	366	ADD A, CCAP3L
013E	F5ED	367	MOV CCAP3L, A
0140	7402	368	MOV A, #FULL BIT HIGH
0142	35FD	369	ADDC A, CCAP3H
0144	F5FD	370	MOV CCAP3H, A
0146	D0D0	371	POP PSW
0148	D0E0	372	POP ACC
014A	32	373	RETI
		374	;
		375	;
		376	;
		377	;
		378	;
		379	;
		380	When a byte is received on the full-duplex serial port, it is then
		381	transmitted back to a "dummy" terminal. This terminal checks that
		382	the byte it transmitted to the PCA is the same value it receives back.
		383	;
		384	;
014B	C0E0	384	SERIAL_PORT: PUSH ACC
014D	C0D0	385	PUSH PSW
014F	30980B	386	JNB RI, TXM ; Check whether RI or TI
0152	E599	387	MOV A, SBUF ; caused the interrupt
0154	C298	388	CLR RI
0156	F599	389	MOV SBUF, A
0158	D0D0	390	POP PSW
015A	D0E0	391	POP ACC
015C	32	392	RETI
		393	;
015D	C299	394	TXM: CLR TI
015F	D0D0	395	POP PSW
0161	D0E0	396	POP ACC
0163	32	397	RETI
		398	;
		399	;
		400	END

REGISTER BANK(S) USED: 0

ASSEMBLY COMPLETE, NO ERRORS FOUND



APPLICATION NOTE

AP-415

July 1988

83C51FA/FB PCA Cookbook

BETSY JONES
ECO APPLICATIONS ENGINEER

This application note illustrates the different functions of the Programmable Counter Array (PCA) which are available on the 83C51FA and 83C51FB. Included are cookbook samples of code in typical applications to simplify the use of the PCA. Since all the examples are written in assembly language, it is assumed the reader is familiar with ASM51. For further information on these products or ASM51 refer to the Embedded Controller Handbook (Vol. I).

PCA OVERVIEW

The major new feature on the 83C51FA and 83C51FB is the Programmable Counter Array. The PCA provides more timing capabilities with less CPU intervention than the standard timer/counters. Its advantages include reduced software overhead and improved accuracy.

The PCA consists of a dedicated timer/counter which serves as the time base for an array of five compare/capture modules. Figure 1 shows a block diagram of the PCA. Notice that the PCA timer and modules are all 16-bits. If an external event is associated with a module, that function is shared with the corresponding Port 1 pin. If the module is not using the port pin, the pin can still be used for standard I/O.

Each of the five modules can be programmed in any one of the following modes:

- Rising and/or Falling Edge Capture
- Software Timer
- High Speed Output
- Watchdog Timer (Module 4 only)
- Pulse Width Modulator.

All of these modes will be discussed later in detail. However, let's first look at how to set up the PCA timer and modules.

PCA TIMER/COUNTER

The timer/counter for the PCA is a free-running 16-bit timer consisting of registers CH and CL (the high and low bytes of the count values). It is the only timer which can service the PCA. The clock input can be selected from the following four modes:

- oscillator frequency \div 12 (Mode 0)
- oscillator frequency \div 4 (Mode 1)
- Timer 0 overflows (Mode 2)
- external input on P1.2 (Mode 3)

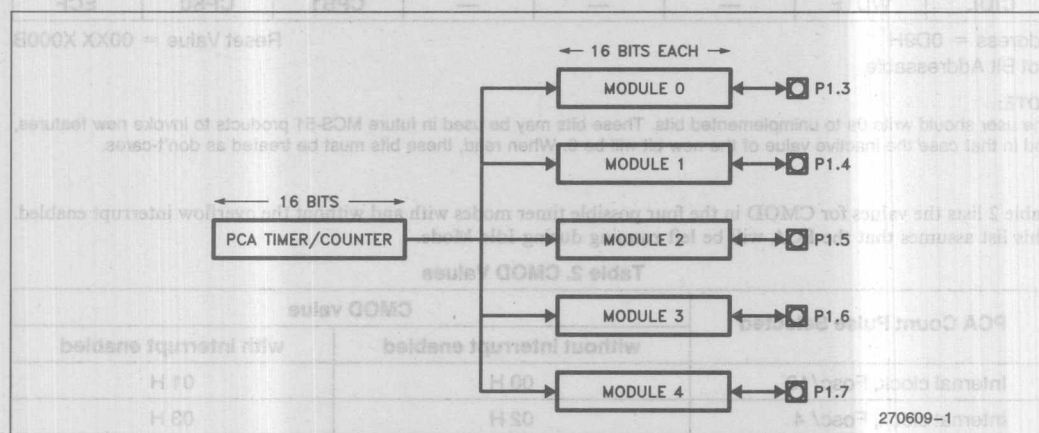


Figure 1. PCA Timer/Counter and Compare/Capture Modules

The table below summarizes the various clock inputs for each mode at two common frequencies. In Mode 0, the clock input is simply a machine cycle count, whereas in Mode 1 the input is clocked three times faster. In Mode 2, Timer 0 overflows are counted allowing for a range of slower inputs to the timer. And finally, if the input is external the PCA timer counts 1-to-0 transitions with the maximum clock frequency equal to $\frac{1}{8}$ x oscillator frequency.

Table 1. PCA Timer/Counter Inputs

PCA Timer/Counter Mode	Clock Increments	
	12 MHz	16 MHz
Mode 0: fosc / 12	1 μ sec	0.75 μ sec
Mode 1: fosc / 4	330 nsec	250 nsec
Mode 2*: Timer 0 Overflows Timer 0 programmed in:		
8-bit mode	256 μ sec	192 μ sec
16-bit mdoe	65 msec	49 msec
8-bit auto-reload	1 to 255 μ sec	0.75 to 191 μ sec
Mode 3: External Input MAX	0.66 μ sec	0.50 μ sec

*In Mode 2, the overflow interrupt for Timer 0 does not need to be enabled.

Special Function Register CMOD contains the Count Pulse Select bits (CPS1 and CPS0) to specify the PCA timer input. This register also contains the ECF bit which enables an interrupt when the counter overflows. In addition, the user has the option of turning off the PCA timer during Idle Mode by setting the Counter Idle bit (CIDL). This can further reduce power consumption by an additional 30%.

CMOD: Counter Mode Register

CIDL	WDTE	—	—	—	CPS1	CPS0	ECF
------	------	---	---	---	------	------	-----

Address = 0D9H

Not Bit Addressable

Reset Value = 00XX X000B

NOTE:

The user should write 0s to unimplemented bits. These bits may be used in future MCS-51 products to invoke new features, and in that case the inactive value of the new bit will be 0. When read, these bits must be treated as don't-cares.

Table 2 lists the values for CMOD in the four possible timer modes with and without the overflow interrupt enabled. This list assumes that the PCA will be left running during Idle Mode.

Table 2. CMOD Values

PCA Count Pulse Selected	CMOD value	
	without interrupt enabled	with interrupt enabled
Internal clock, Fosc/12	00 H	01 H
Internal clock, Fosc/ 4	02 H	03 H
Timer 0 overflow	04H	05 H
External clock at P1.2	06 H	07 H

timer overflows, the Counter Overflow bit (CF) gets set. CCON also contains the five event flags for the PCA modules. The purpose of these flags will be discussed in the next section.

CCON: Counter Control Register

CF	CR	—	CCF4	CCF3	CCF2	CCF1	CCF0
----	----	---	------	------	------	------	------

Address = 0D8H

Reset Value = 00X0 0000B

Bit Addressable

The PCA timer registers (CH and CL) can be read and written to at any time. However, to read the full 16-bit timer value simultaneously requires using one of the PCA modules in the capture mode and toggling a port pin in software. More information on reading the PCA timer is provided in the section on the Capture Mode.

CCF0	CCF1	CCF2	CCF3	CCF4	—	CH	CF
------	------	------	------	------	---	----	----

COMPARE/CAPTURE MODULES

Each of the five compare/capture modules has a mode register called CCAPMn (n = 0,1,2,3,or 4) to select which function it will perform. Note the ECCFn bit which enables an interrupt to occur when a module's event flag is set.

CCAPMn: Compare/Capture Mode Register

—	ECOMn	CAPPn	CAPNn	MATn	TOGn	PWMn	ECCFn
---	-------	-------	-------	------	------	------	-------

Address = 0DAH (n = 0)

Reset Value = X000 0000B

0DBH (n = 1)

0DCH (n = 2)

0DDH (n = 3)

0DEH (n = 4)

Table 3 lists the CCAPMn values for each different mode with and without the PCA interrupt enabled; that is, the interrupt is optional for all modes. However, some of the PCA modes require software servicing. For example, the Capture modes need an interrupt so that back-to-back events can be recognized. Also, in most applications the purpose of the Software Timer mode is to generate interrupts in software so it would be useless not to have the interrupt enabled. The PWM mode, on the other hand, does not require CPU intervention so the interrupt is normally not enabled.

Table 3. Compare/Capture Mode Values

Module Function	CCAPMn Value	
	without interrupt enabled	with interrupt enabled
Capture Positive only	20H	21 H
Capture Negative only	10H	11 H
Capture Pos. or Neg.	30H	31 H
Software Timer	48H	49 H
High Speed Output	4C H	4D H
Watchdog Timer	48 or 4C H	—
Pulse Width Modulator	42 H	43H

It should be mentioned that a particular module can change modes within the program. For example, a module might be used to sample incoming data. Initially it could be set up to capture a falling edge transition. Then the same module can be reconfigured as a software timer to interrupt the CPU at regular intervals and sample the pin.

Each module also has a pair of 8-bit compare/capture registers (CCAPnH, CCAPnL) associated with it. These registers are used to store the time when a capture event occurred or when a compare event should occur. Remember, event times are based on the free-running PCA timer (CH and CL). For the PWM mode, the high byte register CCAPnH controls the duty cycle of the waveform.

When an event occurs, a flag in CCON is set for the appropriate module. This register is bit addressable so that event flags can be checked individually.

CCON: Counter Control Register

CF	CR	—	CCF4	CCF3	CCF2	CCF1	CCF0
----	----	---	------	------	------	------	------

Address = 0D8H

Bit Addressable

These five event flags plus the PCA timer overflow flag share an interrupt vector as shown below. These flags are not cleared when the hardware vectors to the PCA interrupt address (0033H) so that the user can determine which event caused the interrupt. This also allows the user to define the priority of servicing each module.

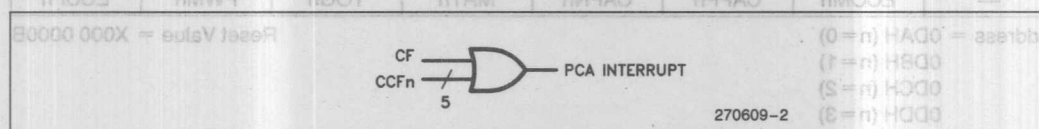


Figure 2. PCA Interrupt

An additional bit was added to the Interrupt Enable (IE) register for the PCA interrupt. Similarly, a high priority bit was added to the Interrupt Priority (IP) register.

IE: Interrupt Enable Register

EA	EC	ET2	ES	ET1	EX1	ET0	EX0
----	----	-----	----	-----	-----	-----	-----

Address = 0A8H

Bit Addressable

Reset Value = 0000 0000B

IP: Interrupt Priority Register

—	PPC	PT2	PS	PT1	PX1	PT0	PX0
---	-----	-----	----	-----	-----	-----	-----

Address = 0B8H

Bit Addressable

Reset Value = X000 0000B

Remember, each of the six possible sources for the PCA interrupt must be individually enabled as well—in the CCAPMn register for the modules and in the CCON register for the timer.

CAPTURE MODE

Both positive and negative transitions can trigger a capture with the PCA. This allows the PCA flexibility to measure periods, pulse widths, duty cycles, and phase differences on up to five separate inputs. This section gives examples of all these different applications.

Figure 3 shows how the PCA handles a capture event. Using Module 0 for this example, the signal is input to P1.3. When a transition is detected on that pin, the 16-bit value of the PCA timer (CH,CL) is loaded into the capture registers (CCAP0H,CCAP0L). Module 0's event flag is set and an interrupt is flagged. The interrupt will then be generated if it has been properly enabled.

In the interrupt service routine, the 16-bit capture value must be saved in RAM before the next event capture occurs; a subsequent capture will write over the first capture value. Also, since the hardware does not clear the event flag, it must be cleared in software.

The time it takes to service this interrupt routine determines the resolution of back-to-back events with the same PCA module. To store two 8-bit registers and clear the event flag takes at least 9 machine cycles. That includes the call to the interrupt routine. At 12 MHz, this routine would take less than 10 microseconds. However, depending on the frequency and interrupt latency, the resolution will vary with each application.

Measuring Pulse Widths

To measure the pulse width of a signal, the PCA module must capture both rising and falling edges (see Figure 4). The module can be programmed to capture either edge if it is known which edge will occur first. However, if this is not known, the user can select which edge will trigger the first capture by choosing the proper mode for the module.

Listing 1 shows an example of measuring pulse widths. (It's assumed the incoming signal matches the one in Figure 4.) In the interrupt routine the first set of capture values are stored in RAM. After the second capture, a subtraction routine calculates the pulse width in units of PCA timer ticks. Note that the subtraction does not have to be completed in the interrupt service routine. Also, this example assumes that the two capture events will occur within 2^{16} counts of the PCA timer, i.e. rollovers of the PCA timer are not counted.

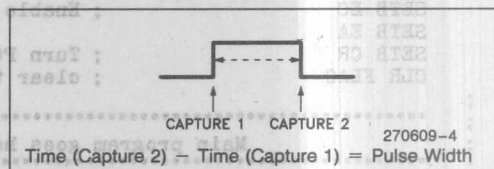


Figure 4. Measuring Pulse Width

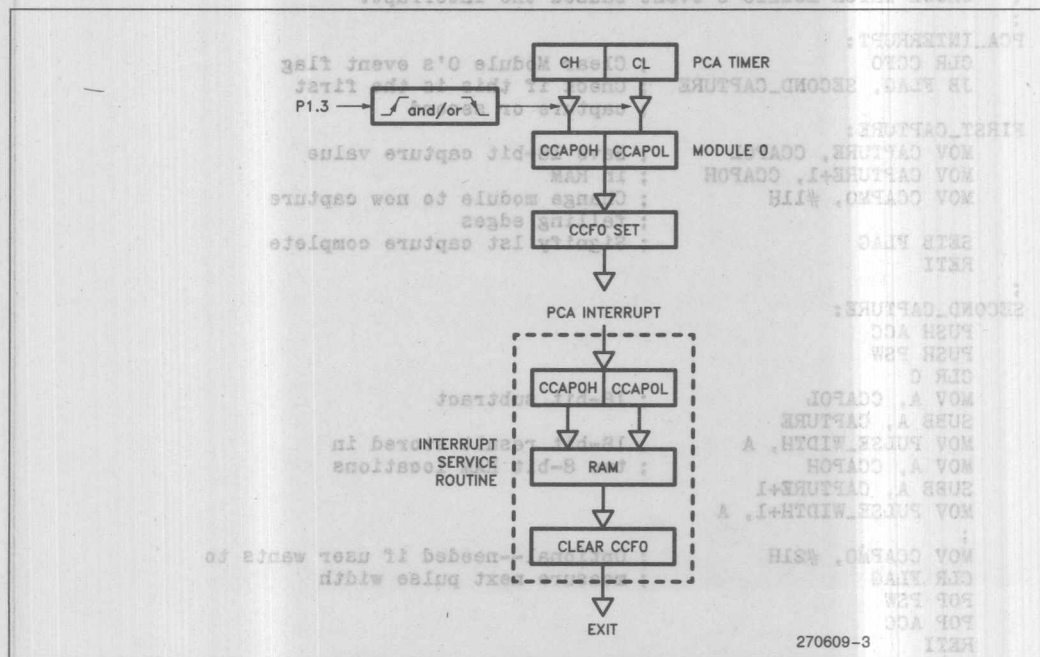


Figure 3. PCA Capture Mode (Module 0)

Listing 1. Measuring Pulse Widths

```

; RAM locations to store capture values
CAPTURE      DATA      30H
PULSE_WIDTH  DATA      32H
FLAG         BIT         20H.0
;
ORG 0000H
JMP PCA_INIT
ORG 0033H
JMP PCA_INTERRUPT
;
PCA_INIT:
MOV CMOD, #00H      ; Initialize PCA timer
MOV CH, #00H        ; Input to timer = 1/12 X Fosc
MOV CL, #00H
;
; Initialize Module 0 in capture mode
MOV CCAPMO, #21H    ; Capture positive edge first
; for measuring pulse width
;
SETB EC            ; Enable PCA interrupt
SETB EA
SETB CR            ; Turn PCA timer on
CLR FLAG           ; clear test flag
;
; *****
; Main program goes here
; *****
;
; This example assumes Module 0 is the only PCA module
; being used. If other modules are used, software must
; check which module's event caused the interrupt.
;
PCA_INTERRUPT:
CLR CCFO           ; Clear Module 0's event flag
JB FLAG, SECOND_CAPTURE ; Check if this is the first
; capture or second
;
FIRST_CAPTURE:
MOV CAPTURE, CCAPOL ; Save 16-bit capture value
MOV CAPTURE+1, CCAPOH ; in RAM
MOV CCAPMO, #11H    ; Change module to now capture
; falling edges
SETB FLAG           ; Signify 1st capture complete
RETI
;
SECOND_CAPTURE:
PUSH ACC
PUSH PSW
CLR C
MOV A, CCAPOL
SUBB A, CAPTURE     ; 16-bit subtract
MOV PULSE_WIDTH, A ; 16-bit result stored in
MOV A, CCAPOH        ; two 8-bit RAM locations
SUBB A, CAPTURE+1
MOV PULSE_WIDTH+1, A
;
MOV CCAPMO, #21H    ; Optional--needed if user wants to
CLR FLAG            ; measure next pulse width
POP PSW
POP ACC
RETI

```


Measuring Periods

Measuring the period of a signal with the PCA is similar to measuring the pulse width. The only difference will be the trigger source for the capture mode. In Figure 5, rising edges are captured to calculate the period. The code is identical to Listing 1 except that the capture mode should not be changed in the interrupt routine. The result of the subtraction will be the period.

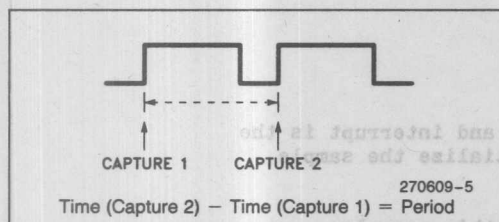


Figure 5. Measuring Period

Measuring Frequencies

Measuring a frequency with the PCA capture mode involves calculating a sample time for a known number of samples. In Figure 6, the time between the first capture and the "Nth" capture equals the sample time T. Listing 2 shows the code for N = 10 samples. It's assumed that the sample time is less than 2¹⁶ counts of the PCA timer.

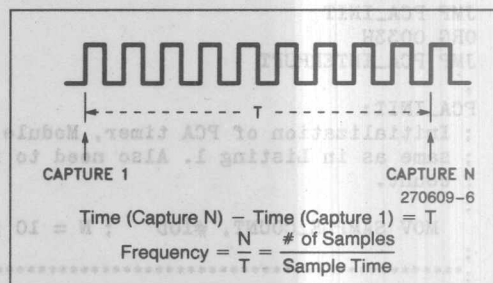


Figure 6. Measuring Frequency

Listing 2. Measuring Frequencies

```

; RAM locations to store capture values
CAPTURE DATA 30H
PERIOD DATA 32H
SAMPLE_COUNT DATA 34H
FLAG BIT 20H.0

ORG 0000H
JMP PCA_INIT
ORG 0033H
JMP PCA_INTERRUPT
;
PCA_INIT:
; Initialization of PCA timer, Module 0, and interrupt is the
; same as in Listing 1. Also need to initialize the sample
; count.
MOV SAMPLE_COUNT, #10D ; N = 10 for this example
;
; *****
; Main program goes here
; *****
; This code assumes only Module 0 is being used.
PCA_INTERRUPT:
CLR CCFO ; Clear module 0's event flag
JB FLAG, NEXT_CAPTURE
;
FIRST_CAPTURE:
MOV CAPTURE, CCAPOH
MOV CAPTURE+1, CCAPOH
SETB FLAG ; Signify first capture complete
RETI
;
NEXT_CAPTURE:
DJNZ SAMPLE_COUNT, EXIT
PUSH ACC
PUSH PSW
CLR C
MOV A, CCAPOH ; 16-bit subtraction
SUBB A, CAPTURE
MOV PERIOD, A
MOV A, CCAPOH
SUBB A, CAPTURE+1
MOV PERIOD+1, A
;
MOV SAMPLE_COUNT, #10D ; Reload for next period
CLR FLAG
POP PSW
POP ACC
EXIT:
RETI

```

The user may instead want to measure frequency by counting pulses for a known sample time. In this case, one module is programmed in the capture mode to count edges (either rising or falling), and a second module is programmed as a software timer to mark the sample time. An example of a software timer is given later. For information on resolution in measuring frequencies, refer to Article Reprint AR-517, "Using the 8051 Microcontroller with Resonant Transducers," in the Embedded Controller Handbook.

Measuring Duty Cycles

To measure the duty cycle of an incoming signal, both rising and falling edges need to be captured. Then the duty cycle must be calculated based on three capture values as seen in Figure 7. The same initialization routine is used from the previous example. Only the PCA interrupt service routine is given in Listing 3.

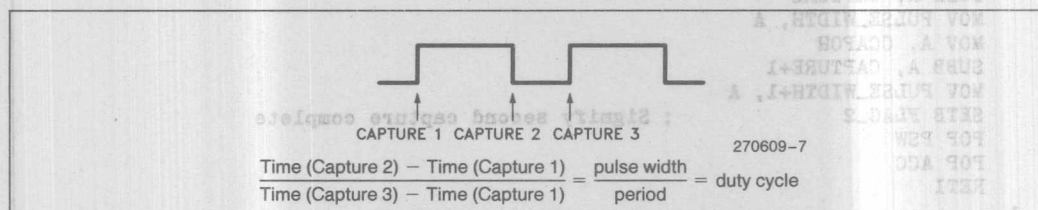


Figure 7. Measuring Duty Cycle

Listing 3. Measuring Duty Cycle

```

; RAM locations to store capture values
CAPTURE      DATA    30H
PULSE_WIDTH   DATA    32H
PERIOD        DATA    34H
FLAG_1        BIT     20H.0
FLAG_2        BIT     20H.1

;
ORG 0000H
JMP PCA_INIT
ORG 0033H
JMP PCA_INTERRUPT

PCA_INIT:
; Initialization for PCA timer, module, and interrupt the same
; as in Listing 1. Capture positive edge first, then either
; edge.
;
; *****
; Main program goes here
; *****
;
; This code assumes only Module 0 is being used.
PCA_INTERRUPT:
CLR CCFO      ; Clear Module 0's event flag
JB FLAG_1, SECOND_CAPTURE
;
FIRST_CAPTURE:
MOV CAPTURE, CCAPOL
MOV CAPTURE+1, CCAPOH
SETB FLAG_1   ; Signify first capture complete
MOV CCAPMO, #31H ; Capture either edge now
RETI

```

Listing 3. Measuring Duty Cycle (Continued)

```

; SECOND_CAPTURE:
PUSH ACC
PUSH PSW
JB FLAG_2, THIRD_CAPTURE
CLR C                                ; Calculate pulse width
MOV A, CCAPOL                       ; 16-bit subtract
SUBB A, CAPTURE
MOV PULSE_WIDTH, A
MOV A, CCAPOH
SUBB A, CAPTURE+1
MOV PULSE_WIDTH+1, A
SETB FLAG_2                         ; Signify second capture complete
POP PSW
POP ACC
RETI

; THIRD_CAPTURE:
CLR C                                ; Calculate period
MOV A, CCAPOL                       ; 16-bit subtract
SUBB A, CAPTURE
MOV PERIOD, A
MOV A, CCAPOH
SUBB A, CAPTURE+1
MOV PERIOD+1, A
MOV CCAPMO, #21H                   ; Optional - reconfigure module to
CLR FLAG_1                         ; capture positive edges for next
CLR FLAG_2                         ; cycle
POP PSW
POP ACC
RETI

```

After the third capture, a 16-bit by 16-bit divide routine needs to be executed. This routine is located in Appendix B. Due to its length, it's up to the user whether the divide routine should be completed in the interrupt routine or be called as a subroutine from the main program.

between two or more signals. For this example, two signals are input to Modules 0 and 1 as seen in Figure 8. Both modules are programmed to capture rising edges only. Listing 4 shows the code needed to measure the difference between these two signals. This code does not assume one signal is leading or lagging the other.

Measuring Phase Differences

Because the PCA modules share the same time base, the PCA is useful for measuring the phase difference

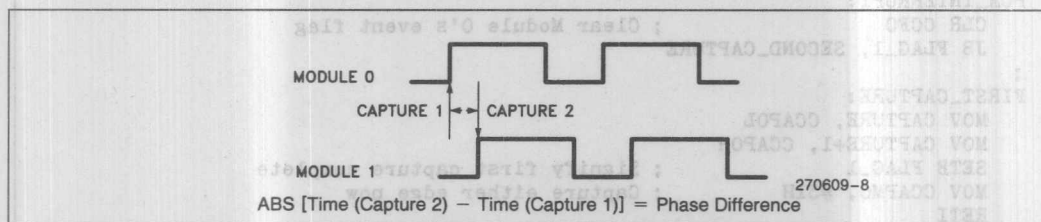


Figure 8. Measuring Phase Differences

Listing 4. Measuring Phase Differences

```

; RAM locations to store capture values
CAPTURE_0 DATA 30H ;
CAPTURE_1 DATA 32H ;
PHASE DATA 34H ;
FLAG_0 BIT 20H.0 ;
FLAG_1 BIT 20H.1 ;

;
ORG 0000H
JMP PCA_INIT
ORG 0033H
JMP PCA_INTERRUPT
;
PCA_INIT:
; Same initialization for PCA timer, and interrupt as
; in Listing 1. Initialize two PCA modules as follows:
;
MOV CCAPM0, #21H ; Module 0 capture rising edges
MOV CCAPM1, #21H ; Module 1 same

;
; *****
; Main program goes here
; *****
; This code assumes only Modules 0 and 1 are being used.
PCA_INTERRUPT:
JB CCFO, MODULE_0 ; Determine which module's
JB CCF1, MODULE_1 ; event caused the interrupt
;
MODULE_0:
CLR CCFO ; Clear Module 0's event flag
MOV CAPTURE_0, CCAPOL ; Save 16-bit capture value
MOV CAPTURE_0+1, CCAPOH
JB FLAG_1, CALCULATE_PHASE ; If capture complete on Module 1, go to calculation
; Module 1, go to calculation
SETB FLAG_0 ; Signify capture on Module 0
RETI

```

Listing 4. Measuring Phase Differences (Continued)

```

MODULE_1:
    CLR CCF_1                ; Clear Module 1's event flag
    MOV CAPTURE_1, CCAP1L
    MOV CAPTURE_1+1, CCAP1H
    JB FLAG_0, CALCULATE_PHASE ; If capture complete on
                                ; Module 0, go to calculation
    SETB FLAG_1              ; Signify capture on Module 1
    RETI

;
; CALCULATE_PHASE:
    PUSH ACC                ; This calculation does not
    PUSH PSW                ; have to be completed in the
    CLR C                   ; interrupt service routine
    ;
    JB FLAG_0, MOD0_LEADING
    JB FLAG_1, MOD1_LEADING
    ;
MOD0_LEADING:
    MOV A, CAPTURE_1
    SUBB A, CAPTURE_0
    MOV PHASE, A
    MOV A, CAPTURE_1+1
    SUBB A, CAPTURE_0+1
    MOV PHASE+1, A
    CLR FLAG_0
    JMP EXIT
    ;
MOD1_LEADING:
    MOV A, CAPTURE_0
    SUBB A, CAPTURE_1
    MOV PHASE, A
    MOV A, CAPTURE_0+1
    SUBB A, CAPTURE_1+1
    MOV PHASE+1, A
    CLR FLAG_1
    RETI

EXIT:
    POP PSW
    POP ACC
    RETI

```

Reading the PCA Timer

Some applications may require that the PCA timer be read instantaneously as a real-time event. Since the timer consists of two 8-bit registers (CH,CL), it would normally take two MOV instructions to read the whole timer. An invalid read could occur if the registers rolled over in the middle of the two MOVs.

However, with the capture mode a 16-bit timer value can be loaded into the capture registers by toggling a port pin. For example, configure Module 0 to capture falling edges and initialize P1.3 to be high. Then when the user wants to read the PCA timer, clear P1.3 and the full 16-bit timer value will be saved in the capture registers. It's still optional whether the user wants to generate an interrupt with the capture.

COMPARE MODE

In this mode, the 16-bit value of the PCA timer is compared with a 16-bit value pre-loaded in the module's compare registers. The comparison occurs three times per machine cycle in order to recognize the fastest possible clock input, i.e. $\frac{1}{4} \times$ oscillator frequency. When there is a match, one of three events can happen:

- (1) an interrupt — Software Timer mode
- (2) toggle of a port pin — High Speed Output mode
- (3) a reset — Watchdog Timer mode.

Examples of each compare mode will follow.

SOFTWARE TIMER

In most applications a software timer is used to trigger interrupt routines which must occur at periodic intervals. Figure 9 shows the sequence of events for the Software Timer mode. The user preloads a 16-bit value in a module's compare registers. When a match occurs between this compare value and the PCA timer, an event flag is set and an interrupt is flagged. An interrupt is then generated if it has been enabled.

If necessary, a new 16-bit compare value can be loaded into (CCAP0H, CCAP0L) during the interrupt routine. The user should be aware that the hardware temporarily disables the comparator function while these registers are being updated so that an invalid match will not occur. That is, a write to the low byte (CCAPn0) disables the comparator while a write to the high byte (CCAP0H) re-enables the comparator. For this reason, user software must write to CCAP0L first, then CCAP0H. The user may also want to hold off any interrupts from occurring while these registers are being updated. This can easily be done by clearing the EA bit. See the code example in Listing 5.

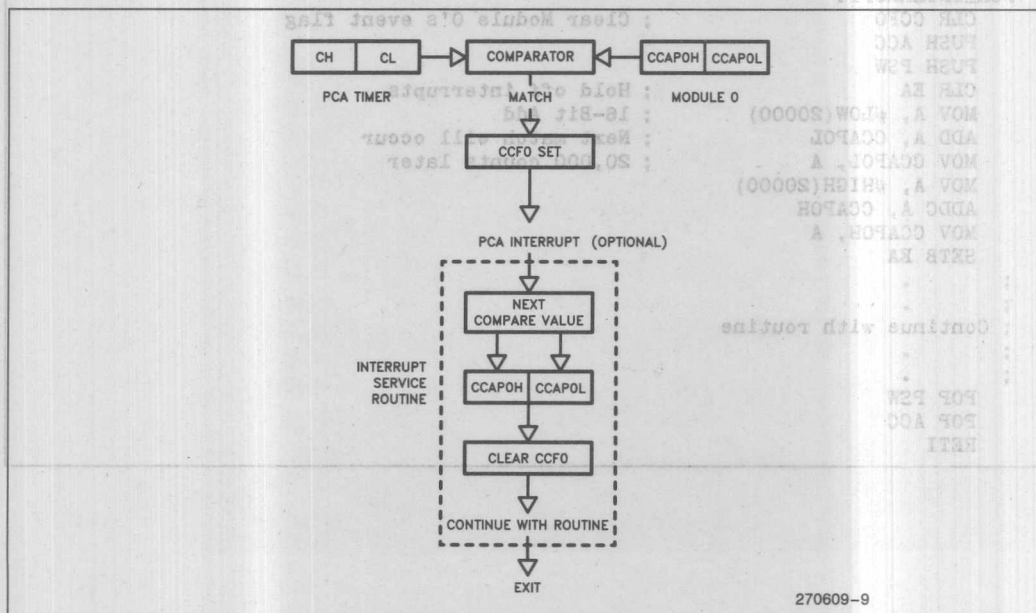


Figure 9. Software Timer Mode (Module 0)

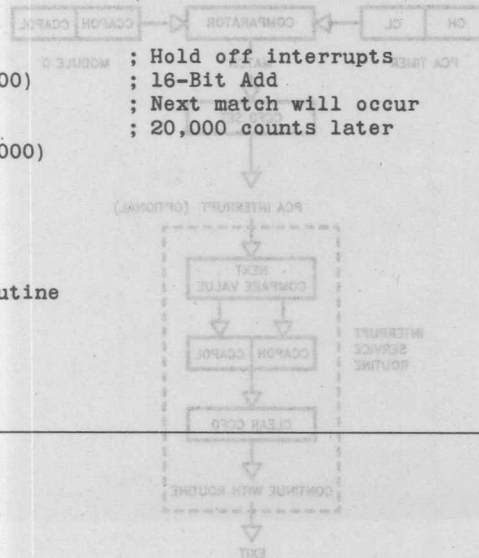
```

; Generate an interrupt in software every 20 msec
;
; Frequency      = 12 MHz
; PCA clock input = 1/12 x Fosc → 1 μsec
; Calculate reload value for compare registers:
;      20 msec
;      ----- = 20,000 counts
;      1 μsec/count
;
ORG 0000H
JMP PCA_INIT
ORG 0033H
JMP PCA_INTERRUPT

PCA_INIT:
; Initialize PCA timer same as in Listing 1
; MOV CCAPM0, #49H      ; Module 0 in Software Timer mode
; MOV CCAPOL, #LOW(20000) ; Write to low byte first
; MOV CCAPH0, #HIGH(20000)
;
; SETB EC              ; Enable PCA interrupt
; SETB EA
; SETB CR              ; Turn on PCA timer
;
; *****
; Main program goes here
; *****

PCA_INTERRUPT:
; Clear Module 0's event flag
CLR CCFO
PUSH ACC
PUSH PSW
CLR EA
MOV A, #LOW(20000)
ADD A, CCAPOL
MOV CCAPOL, A
MOV A, #HIGH(20000)
ADDC A, CCAPH0
MOV CCAPH0, A
SETB EA
;
; Continue with routine
;
POP PSW
POP ACC
RETI

```



HIGH SPEED OUTPUT

The High Speed Output (HSO) mode toggles a port pin when a match occurs between the PCA timer and the pre-loaded value in the compare registers (see Figure 10). The HSO mode is more accurate than toggling pins in software because the toggle occurs *before* branching to an interrupt, i.e. interrupt latency will not effect the accuracy of the output. In fact, the interrupt is optional. Only if the user wants to change the time for the next toggle is it necessary to update the compare registers. Otherwise, the next toggle will occur when the PCA timer rolls over and matches the last compare value. Examples of both are shown.

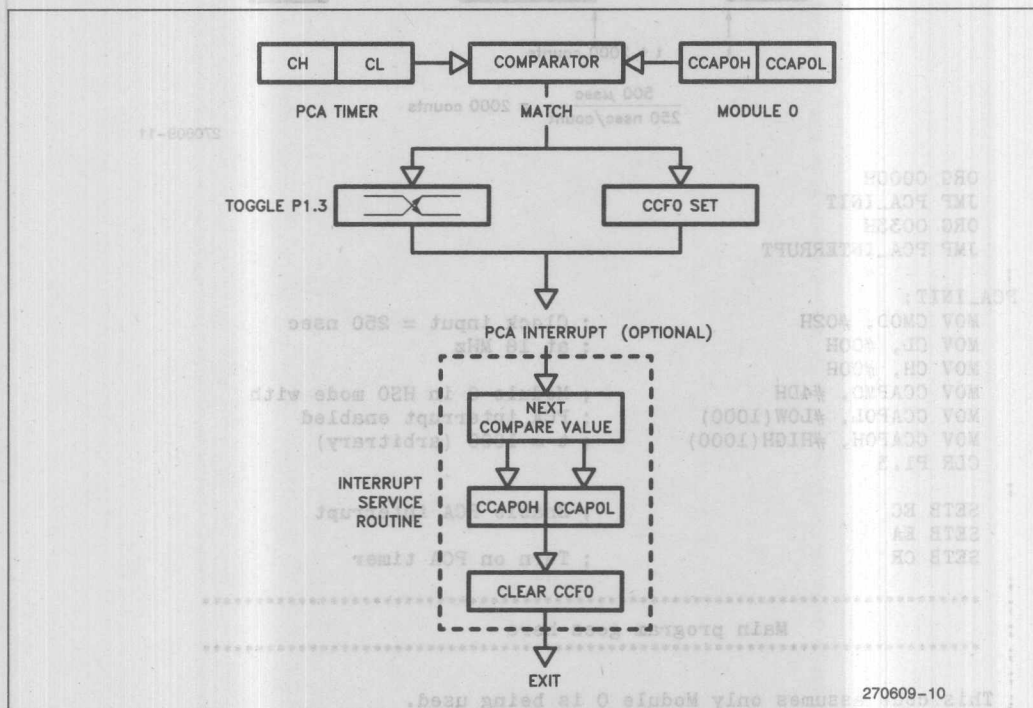


Figure 10. High Speed Output Mode (Module 0)

Without any CPU intervention, the fastest waveform the PCA can generate with the HSO mode is a 30.5 Hz signal at 16 MHz. Refer to Listing 6. By changing the PCA clock input, slower waveforms can also be generated.

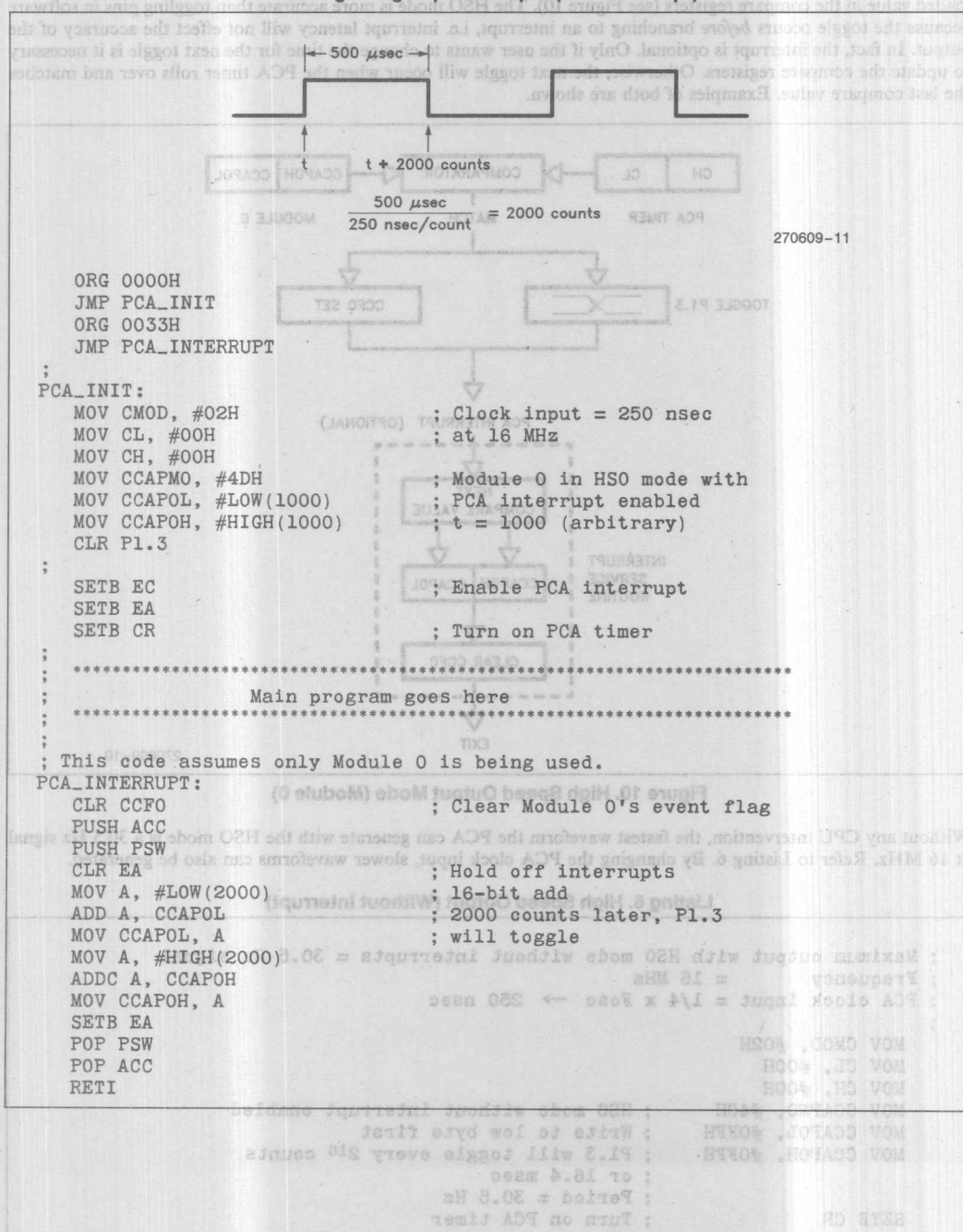
Listing 6. High Speed Output (Without Interrupt)

```

; Maximum output with HSO mode without interrupts = 30.5 Hz signal
; Frequency = 16 MHz
; PCA clock input = 1/4 x Fosc -> 250 nsec
;
MOV CMOD, #02H
MOV CL, #00H
MOV CH, #00H
MOV CCAPMO, #4CH ; HSO mode without interrupt enabled
MOV CCAPOL, #0FFH ; Write to low byte first
MOV CCAP0H, #0FFH ; P1.3 will toggle every 2^16 counts
; or 16.4 msec
; Period = 30.5 Hz
; Turn on PCA timer
SETB CR
  
```

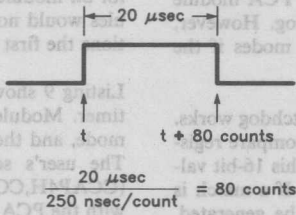
In this next example, the PCA interrupt is used to change the compare value for each toggle. This way a variable frequency output can be generated. Listing 7 shows an output of 1 KHz at 16 Mhz.

Listing 7. High Speed Output (With Interrupt)



Another option with the HSO mode is to generate a single pulse. Listing 8 shows the code for an output with a pulse width of 20 μsec . As in the previous example, the PCA interrupt will be used to change the time for the toggle. The first toggle will occur at time "t". After 80 counts of the PCA timer, 20 μsec will have expired, and the next toggle will occur. Then the HSO mode will be disabled.

Listing 8. High Speed Output (Single Pulse)



```

ORG 0000H
JMP PCA_INIT
ORG 0033H
JMP PCA_INTERRUPT

PCA_INIT:
MOV CMOD, #02H      ; Clock input = 250 nsec
MOV CL, #00H        ; at 16 Mhz
MOV CH, #00H
MOV CCAPMO, #4DH     ; Module 0 in HSO mode with
MOV CCAPOL, #LOW(1000) ; PCA interrupt enabled
MOV CCAPOH, #HIGH(1000) ; t = 1000 (arbitrary)
CLR P1.3

;
SETB EC              ; Enable PCA interrupt
SETB EA
SETB CR              ; Turn on PCA timer

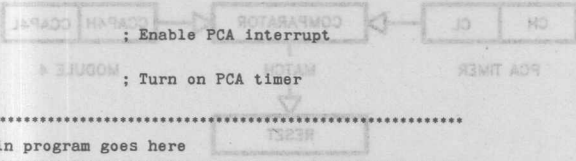
;
; .....
; Main program goes here
; .....

; This code assumes only Module 0 is being used.
PCA_INTERRUPT:
CLR CCFO             ; Clear Module 0's event flag
JNB P1.3, DONE

;
PUSH ACC
PUSH PSW
CLR EA              ; Hold off interrupts
MOV A, #LOW(80)     ; 16-bit add
ADD A, CCAPOL        ; 80 counts later, P1.3
MOV CCAPOL, A        ; will toggle
MOV A, #HIGH(80)
ADDC A, CCAPOH
MOV CCAPOH, A
SETB EA
POP PSW
POP ACC
RETI

;
DONE:
MOV CCAPMO, #00H     ; Disable HSO mode
RETI

```



WATCHDOG TIMER

An on-board watchdog timer is available with the PCA to improve the reliability of the system without increasing chip count. Watchdog timers are useful for systems which are susceptible to noise, power glitches, or electrostatic discharge. Module 4 is the only PCA module which can be programmed as a watchdog. However, this module can still be used for other modes if the watchdog is not needed.

Figure 11 shows a diagram of how the watchdog works. The user pre-loads a 16-bit value in the compare registers. Just like the other compare modes, this 16-bit value is compared to the PCA timer value. If a match is allowed to occur, an internal reset will be generated. This will not cause the RST pin to be driven high.

In order to hold off the reset, the user has three options:

- (1) periodically change the compare value so it will never match the PCA timer,
- (2) periodically change the PCA timer value so it will never match the compare value, or
- (3) disable the watchdog by clearing the WDTE bit before a match occurs and then re-enable it.

The first two options are more reliable because the watchdog timer is never disabled as in option #3. If the program counter ever goes astray, a match will eventually occur and cause an internal reset. The second option is also not recommended if other PCA modules are being used. Remember, the PCA timer is the time base for *all* modules; changing the time base for other modules would not be a good idea. Thus, in most applications the first solution is the best option.

Listing 9 shows the code for initializing the watchdog timer. Module 4 can be configured in either compare mode, and the WDTE bit in CMOD must also be set. The user's software then must periodically change (CCAP4H,CCAP4L) to keep a match from occurring with the PCA timer (CH,CL). This code is given in the WATCHDOG routine.

This routine should not be part of an interrupt service routine. Why? Because if the program counter goes astray and gets stuck in an infinite loop, interrupts will still be serviced and the watchdog will keep getting reset. Thus, the purpose of the watchdog would be defeated. Instead call this subroutine from the main program within 2^{16} count of the PCA timer.

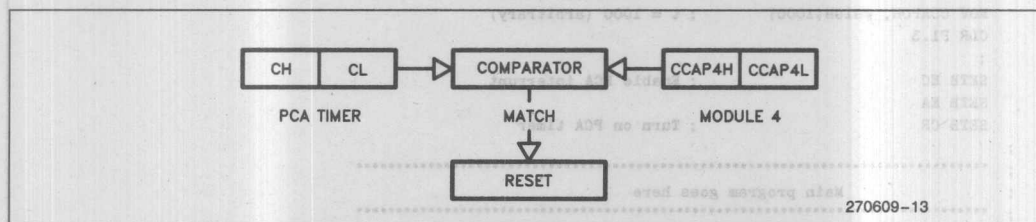


Figure 11. Watchdog Timer Mode (Module 4)

Listing 9. Watchdog Timer

```

INIT_WATCHDOG:
    MOV CCAPM4, #4CH      ; Module 4 in compare mode
    MOV CCAP4L, #0FFH     ; Write to low byte first
    MOV CCAP4H, #0FFH     ; Before PCA timer counts up to
                          ; FFFF Hex, these compare values
                          ; must be changed
    ORL CMOD, #40H        ; Set the WDTE bit to enable the
                          ; watchdog timer without changing
                          ; the other bits in CMOD
;
; *****
; Main program goes here, but CALL WATCHDOG periodically.
; *****
;
WATCHDOG:
    CLR EA                ; Hold off interrupts
    MOV CCAP4L, #00       ; Next compare value is within
    MOV CCAP4H, CH         ; 255 counts of the current PCA
    SETB EA                ; timer value
    RET
    
```

PULSE WIDTH MODULATOR

The PCA can generate 8-bit PWMs by comparing the low byte of the PCA timer (CL) with the low byte of the compare registers (CCAPnL). When $CL < CCAPnL$ the output is low. When $CL \geq CCAPnL$ the output is high.

To control the duty cycle of the output, the user actually loads a value into the high byte CCAPnH (see Figure 12). Since a write to this register is asynchronous, a new value is not shifted into CCAPnL for comparison until

the next period of the output: that is, when CL rolls over from 255 to 00. This mechanism provides “glitch-free” writes to CCAPnH when the duty cycle of the output is changed.

CCAPnH can contain any integer from 0 to 255, but Figure 13 shows a few common duty cycles and the corresponding values for CCAPnH. Note that a 0% duty cycle can be obtained by writing to the port pin directly with the CLR bit instruction. To calculate the CCAPnH value for a given duty cycle, use the following equation:

$$CCAPnH = 256 (1 - \text{Duty Cycle})$$

where CCAPnH is an 8-bit integer and Duty Cycle is expressed as a fraction.

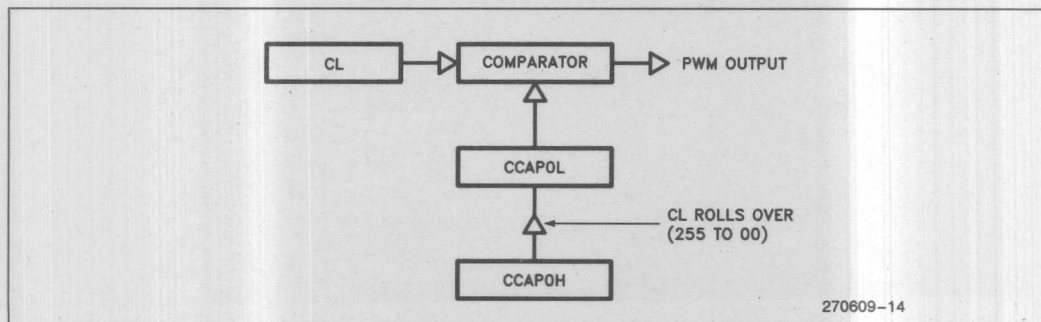


Figure 12. PWM Mode (Module 0)

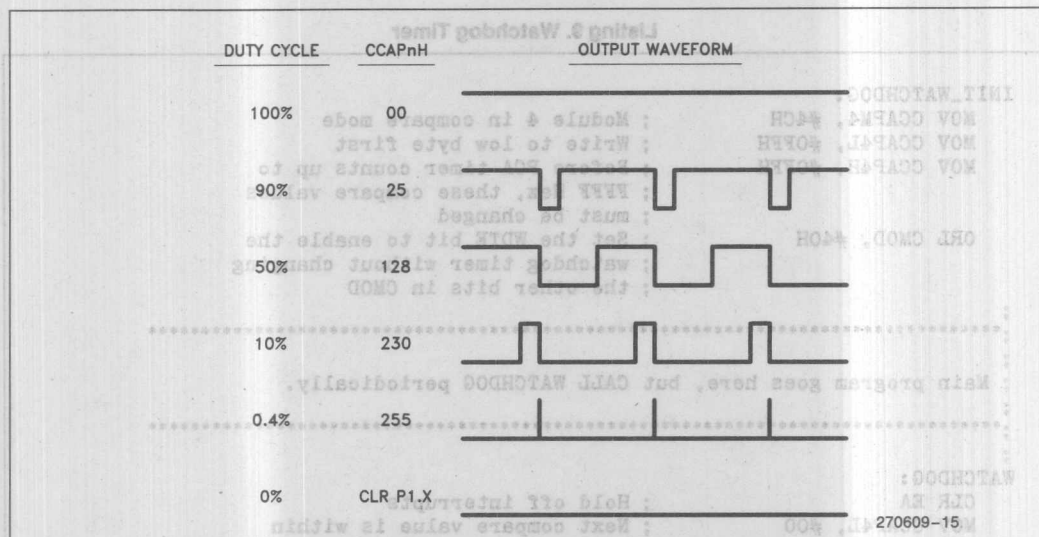


Figure 13. CCAPnH Varies Duty Cycle

Table 4. PWM Frequencies.

PCA Timer Mode	PWM Frequency	
	12 MHz	16 MHz
1/12 Osc. Frequency	3.9 KHz	5.2 KHz
1/4 Osc. Frequency	11.8 KHz	15.6 KHz
Timer 0 Overflow:		
8-bit	15.5 Hz	20.3 Hz
16-bit	0.06 Hz	0.08 Hz
8-bit Auto-Reload	3.9 KHz to 15.3 Hz	5.2 KHz to 20.3 Hz
External Input (Max)	5.9 KHz	7.8 KHz

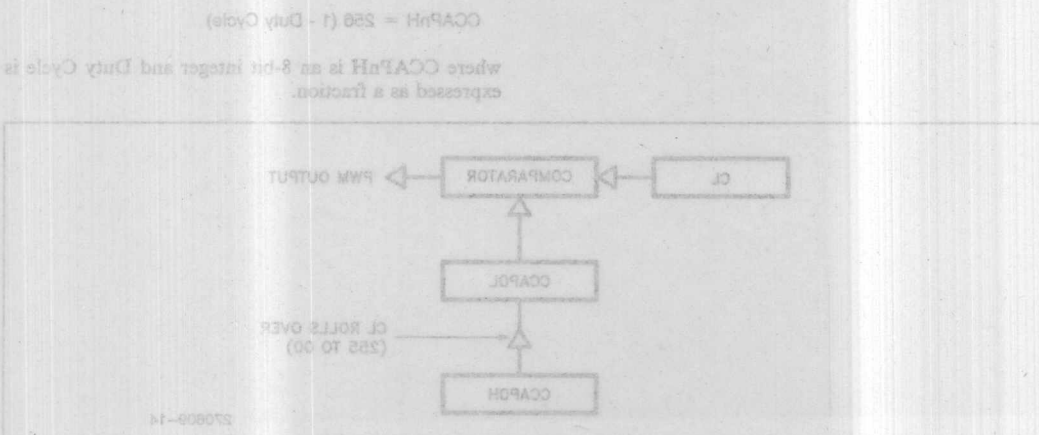


Figure 12. PWM Mode (Module 0)

Listing 10. PWM

```

INIT-PWM:
MOV CMOD, #02H      ; Clock input = 250 nsec at 16 MHz
MOV CL, #00H        ; Frequency of output = 15.6 KHz
MOV CH, #00H
MOV CCAPM0, #42H    ; Module 0 in PWM mode
MOV CCAPOL, #00H
MOV CCAPOH, #128D   ; 50 percent duty cycle
;
SETB CR             ; Turn on PCA timer

```

The frequency of the PWM output will depend on which of the four inputs is chosen for the PCA timer. The maximum frequency is 15.6 KHz at 16 MHz. Refer to Table 4 for a summary of the different PWM frequencies possible with the PCA.

Listing 10 shows how to initialize Module 0 for a PWM signal at 50% duty cycle. Notice that no PCA interrupt is needed to generate the PWM (i.e. no software overhead!). To create a PWM output on the 8051 requires a hardware timer plus software overhead to toggle the port pin. The advantage of the PCA is obvious, not to mention it can support up to 5 PWM outputs with just one chip.

CONCLUSION

This list of examples with the PCA is by no means exhaustive. However, the advantages of the PCA can easily be seen from the given applications. For example, the PCA can provide better resolution than Timers 0, 1 and 2 because the PCA clock rate can be three times faster. The PCA can also perform many tasks that these hardware timers can not, i.e. measure phase differences between signals or generate PWMs. In a sense, the PCA provides the user with five more timer/counters in addition to Timers 0, 1 and 2 on the 8XC51FA/FB.

Appendix A includes test routines for all the software examples in this application note. The divide routine for calculating duty cycles is in Appendix B. And finally, Appendix C is a table of the Special Function Registers for the 8XC51FA/FB with the new or modified registers **boldfaced**.

APPENDIX A TEST ROUTINES

Listing 1a - Measuring Pulse Widths

```

;
; Turn on PCA timer
; 80 percent duty cycle
; Module 0 in PWM mode
;

$nomod51
$nosymbols
$nolist
$include (reg252.pdf)
$list

; Variables
CAPTURE      DATA 30H
PULSE_WIDTH  DATA 32H
FLAG         BIT   20H.0

ORG 0000H
JMP PCA_INIT

ORG 0033H
JMP PCA_INTERRUPT

; Initialize PCA timer
PCA_INIT:    MOV CMOD, #00H      ; Input to PCA timer = 1/12 x Fosc
            MOV CH, #00
            MOV CL, #00

; Initialize Module 0 in capture mode
            MOV CCAPM0, #21H      ; Capture positive edge first on P1.3
            MOV CCAP0H, #00
            MOV CCAP0L, #00

; Enable PCA interrupt
SETB EC
SETB EA
SETB CR
CLR FLAG

; Test program only
WAIT:       JMP $                ; Wait for PCA interrupt
            JMP WAIT

; This code assumes Module 0 is the only module being used. If
; other PCA module's are being used, software must check which
; module's event flag caused the interrupt.

PCA_INTERRUPT:
            CLR CCF0              ; Clear module 0's event flag
            JB FLAG, SECOND_CAPTURE

; FIRST_CAPTURE:
            MOV CAPTURE, CCAP0L
            MOV CAPTURE+1, CCAP0H

```

270609-16


```

MOV CCAPM0, #11H      ; Change module to now capture
                        ; falling edges
SETB FLAG              ; Signify first capture complete
RETI

; SECOND_CAPTURE:
PUSH ACC
PUSH PSW
CLR C
MOV A, CCAP0L          ; 16-bit subtract
SUBB A, CAPTURE
MOV PULSE_WIDTH, A
MOV A, CCAP0H
SUBB A, CAPTURE+1
MOV PULSE_WIDTH+1, A

;
MOV CCAPM0, #21H      ; Optional if user wants to measure
                        ; next pulse width
CLR FLAG
POP PSW
POP ACC
RETI

;
END

```

270609-17

Initializes module 0 in capture mode
 MOV CCAPM0, #21H
 MOV CCAP0H, #00
 MOV CCAP0L, #00
 SETB SC
 SETB SA
 SETB CR
 CLR FLAG
 ; Clear test flag
 ; Turn PCA timer on
 ; Enable PCA interrupt

Test program only
 ; Wait for PCA interrupt
 JNB \$
 JMP WAIT

This code assumes only module 0 is being used. If other modules
 are being used, software must check which module's flag caused
 the interrupt.

PCA_INTERRUPT
 CLR CCR0
 JB FLAG, SECOND_CAPTURE
 FIRST_CAPTURE:
 MOV CAPTURE, CCAP0L
 MOV CAPTURE+1, CCAP0H

270609-18

Listing 1b - Measuring Periods

```

;
;
$nomod51
$nosymbols
$no1ist
$include (reg252.pdf)
$list

```

Variables

```

CAPTURE      DATA      30H
PERIOD        DATA      32H
FLAG          BIT        20H.0

```

```

ORG 0000H
JMP PCA_INIT

```

```

ORG 0033H
JMP PCA_INTERRUPT

```

```

; Initialize PCA timer
PCA_INIT:  MOV CMOD, #00H      ; Input to timer = 1/12 x Fosc
           MOV CH, #00H
           MOV CL, #00

```

```

; Initialize Module 0 in capture mode
MOV CCAPM0, #21H      ; Capture rising edges on P1.3

```

```

MOV CCAP0H, #00
MOV CCAP0L, #00

```

```

SETB EC      ; Enable PCA interrupt
SETB EA
SETB CR      ; Turn PCA timer on
CLR FLAG     ; Clear test flag

```

Test program only

```

WAIT:      JMP $      ; Wait for PCA interrupt
           JMP WAIT

```

```

; This code assumes only Module 0 is being used. If other modules
; are being used, software must check which module's flag caused
; the interrupt.

```

```

PCA_INTERRUPT:
CLR CCF0      ; Clear module 0's event flag
JB FLAG, SECOND_CAPTURE

```

```

FIRST_CAPTURE:
MOV CAPTURE, CCAP0L
MOV CAPTURE+1, CCAP0H

```

270609-18

; 16-Bit subtraction

Listing 2 - Measuring Frequencies

```

:
:
:
$nomod51
$nosymbols
$no1ist
$include (reg252.pdf)
$list
:
:
:
Variables
:
:
CAPTURE          DATA      30H
PERIOD           DATA      32H
SAMPLE_COUNT     DATA      34H
FLAG             BIT        20H.0
:
ORG 0000H
JMP PCA_INIT
:
ORG 0033H
JMP PCA_INTERRUPT
:
:
Initialize PCA timer
PCA_INIT:  MOV CMOD, #00H          ; Input to PCA timer = 1/12 x Fosc
           MOV CH, #00
           MOV CL, #00
:
:
Initialize Module 0 in capture mode
           MOV CCAPM0, #21H        ; Capture positive edges on P1.3
:
           MOV CCAP0H, #00
           MOV CCAP0L, #00
:
           MOV SAMPLE_COUNT, #10D ; N = 10 for this example
:
           SETB EC                 ; Enable PCA interrupt
           SETB EA
           SETB CR                 ; Turn PCA timer on
           CLR FLAG                ; Test flag
:
:
:
Test program only
:
:
WAIT:      JMP $                  ; Wait for PCA interrupt
           JMP WAIT
:
:
:
This code assumes only Module 0 is being used.
:
PCA_INTERRUPT:
           CLR CCF0                ; Clear module 0's event flag
           JB FLAG, NEXT_CAPTURE
:
:

```

270609-20


```

FIRST_CAPTURE:
    MOV CAPTURE, CCAP0L
    MOV CAPTURE+1, CCAP0H
    SETB FLAG_1
    MOV CCAPM0, #31H
    RETI
; Signify first capture complete
; Capture either edge now

; SECOND_CAPTURE:
    PUSH ACC
    PUSH PSW
    JB FLAG_2, THIRD_CAPTURE
    CLR C
    MOV A, CCAP0L
    SUBB A, CAPTURE
    MOV PULSE_WIDTH, A
    MOV A, CCAP0H
    SUBB A, CAPTURE+1
    MOV PULSE_WIDTH+1, A
; Calculate pulse width
; 16-bit subtract

    SETB FLAG_2
    POP PSW
    POP ACC
    RETI
; Signify second capture complete

; THIRD_CAPTURE:
    CLR C
    MOV A, CCAP0L
    SUBB A, CAPTURE
    MOV PERIOD, A
    MOV A, CCAP0H
    SUBB A, CAPTURE+1
    MOV PERIOD+1, A
; Calculate period
; 16-bit subtract

    MOV CCAPM0, #21H
    CLR FLAG_1
    CLR FLAG_2
    POP PSW
    POP ACC
    RETI
; Optional- reconfigure module to
; capture positive edges for
; next cycle

; END

```

270609-23

Listing 4 - Measuring Phase Differences

```

;
;
; $nomod51
; $nosymbols
; $nolist
; $include (reg252.pdf)
; $list
;
; Variables
;
CAPTURE_0      DATA      30H
CAPTURE_1      DATA      32H
PHASE          DATA      34H
;
FLAG_0         BIT        20H.0
FLAG_1         BIT        20H.1
;
ORG 0000H
JMP PCA_INIT
;
ORG 0033H
JMP PCA_INTERRUPT
;
; Initialize PCA timer
PCA_INIT:      MOV CMOD, #00H      ; Input to PCA timer = 1/12 x Fosc
               MOV CH, #00
               MOV CL, #00
;
; Initialize Modules 0 & 1 in capture mode
               MOV CCAPM0, #21H    ; Capture positive edges on P1.3
               MOV CCAP0H, #00
               MOV CCAP0L, #00
;
               MOV CCAPM1, #21H    ; Capture positive edges on P1.4
               MOV CCAP1H, #00
               MOV CCAP1L, #00
;
               MOV R0, #0FFH        ; Used for test program only
               MOV R1, #0FFH
;
               CLR FLAG_0           ; Clear test flags
               CLR FLAG_1
;
               SETB EC              ; Enable PCA interrupt
               SETB EA
               SETB CR              ; Turn PCA timer on
;

```

270609-24


```

; *****
;                               Test program only
; *****
MAIN:      CALL TOG1              ; Generate two waveforms
          CALL DELAY2            ; with known phase difference
          CALL TOG2
          JMP MAIN

; *****
TOG1:      CPL P1.6              ; These two waveforms are input to
          CALL DELAY1            ; P1.3 and P1.4
          RET

; *****
TOG2:      CPL P1.5
          CALL DELAY1
          RET

; *****
DELAY1:    DJNZ R0, $
          RET

; *****
DELAY2:    DJNZ R1, $
          RET
; *****

; This code assumes only Modules 0 and 1 are being used.
; *****
PCA_INTERRUPT:
          JB CCF0, MODULE_0      ; Determine which module's event
          JB CCF1, MODULE_1      ; caused the interrupt

; *****
MODULE_0:
          CLR CCF0              ; Clear Module 0's event flag
          MOV CAPTURE_0, CCAP0L
          MOV CAPTURE_0+1, CCAP0H
          JB FLAG_1, CALCULATE_PHASE ; If capture is complete on Module 1,
          ; go to calculation
          SETB FLAG_0            ; Signify capture complete on
          RETI                  ; Module 0

; *****
MODULE_1:
          CLR CCF1              ; Clear Module 1's event flag
          MOV CAPTURE_1, CCAP1L
          MOV CAPTURE_1+1, CCAP1H
          JB FLAG_0, CALCULATE_PHASE ; If capture is complete on Module 0,
          ; go to calculation
          SETB FLAG_1            ; Signify capture complete
          RETI                  ; Module 1

; *****
CALCULATE_PHASE:
          PUSH ACC              ; This calculation does not have to
          PUSH PSW              ; be completed in the interrupt
          CLR C                  ; service routine

          JB FLAG_0, MOD0_LEADING

```

270609-25

Listing 5. Software Timer

```

;
;
$nomod51
$nosymbols
$nolist
$include (reg252.pdf)
$list
;
; Software Timer mode which interrupts every 20 msec with Fosc = 12 MHz.
;
ORG 0000H
JMP PCA_INIT
ORG 0033H
JMP PCA_INTERRUPT
;
; Initialize PCA timer
PCA_INIT:  MOV CMOD, #00H          ; Input to PCA timer = 1/12 x Fosc
           MOV CH, #00
           MOV CL, #00
;
           MOV CCAPM0, #49H        ; Software Timer mode with interrupt
           MOV CCAP0L, #LOW(20000) ; Write to low byte first
           MOV CCAP0H, #HIGH(20000)
;
           SETB EC                  ; Enable PCA interrupt
           SETB EA
           SETB CR                  ; Turn PCA timer on
;
;
; .....
;
; Test program only
;
WAIT:      JMP $                    ; Wait for PCA interrupt
;
;
; .....
;
; This code assumes Module 0 is the only module being used. If
; other PCA module's are being used, software must check which
; module's event flag caused the interrupt.
;
PCA_INTERRUPT:
           CLR CCF0                ; Clear module 0's event flag
           PUSH ACC
           PUSH PSW
           CLR EA                  ; Hold off interrupts
           MOV A, #LOW(20000)      ; 16-bit add
           ADD A, CCAP0L           ; Next match will occur 20,000
           MOV CCAP0L, A          ; counts later
           MOV A, #HIGH(20000)
           ADDC A, CCAP0H
           MOV CCAP0H, A
           SETB EA
           POP PSW
           POP ACC
           RETI
;
END

```

270609-27

270609-30

270609-32

270609-34

APPENDIX B Duty Cycle Calculation

\$DEBUG

SHORT_DIVISION SEGMENT CODE

EXTRN DATA(PULSE_WIDTH, PERIOD, DUTY_CYCLE)
PUBLIC DUTY_CYCLE_CALCULATION

RSEG SHORT_DIVISION

DUTY_CYCLE_CALCULATION

CALCULATES DUTY_CYCLE = PULSE_WIDTH / PERIOD

Inputs to this routine are 16-bit pulse width and period measurements of a rectangular waveform. The output is a 9-bit BCD number representing the duty cycle of the waveform. The low 8 bits of the result are returned in DUTY_CYCLE. The 9th bit is the carry bit in the PSW. If the duty cycle is between 0 and 99 percent, the carry bit is 0 and DUTY_CYCLE contains the two BCD digits representing the duty cycle as a percent. If the duty cycle is 100 percent, the carry bit is 1 and DUTY_CYCLE contains 0.

INPUTS: PULSE_WIDTH 2 bytes in externally defined DATA
(low byte at PULSE_WIDTH, high byte at PULSE_WIDTH+1)

PERIOD 2 bytes in externally defined DATA
(low byte at PERIOD, high byte at PERIOD+1)

OUTPUT: DUTY_CYCLE 1 byte in externally defined DATA

VARIABLES AND REGISTERS MODIFIED:

PULSE_WIDTH, DUTY_CYCLE
ACC, B, PSW, R2, R3

ERROR EXIT: Exit with OV = 1 indicates PULSE_WIDTH > PERIOD.

DUTY_CYCLE_CALCULATION:

```
MOV A,PERIOD+1
CJNE A,PULSE_WIDTH+1,NOT_EQUAL
MOV A,PERIOD
CJNE A,PULSE_WIDTH,NOT_EQUAL
```

270609-35

```

EQUAL:
    SETB  C
    MOV   DUTY_CYCLE,#0
    CLR   OV
    RET

```

```

NOT_EQUAL:
    JNC   CONTINUE
    SETB  OV
    RET

```

```

CONTINUE:
    MOV   R2,#8
    MOV   DUTY_CYCLE,#0
    MOV   R3,#0

```

```

TIMES_TWO:
    MOV   A,PULSE_WIDTH
    RLC   A
    MOV   PULSE_WIDTH,A
    MOV   A,PULSE_WIDTH+1
    RLC   A
    MOV   PULSE_WIDTH+1,A
    MOV   A,R3
    RLC   A
    MOV   R3,A

```

```

COMPARE:
    CJNE  R3,#0,DONE
    MOV   A,PULSE_WIDTH+1
    CJNE  A,PERIOD+1,DONE
    MOV   A,PULSE_WIDTH
    CJNE  A,PERIOD,DONE

```

```

DONE:
    CPL   C

```

```

BUILD_DUTY_CYCLE:
    MOV   A,DUTY_CYCLE
    RLC   A
    MOV   DUTY_CYCLE,A
    JNB   ACC.0,LOOP_CONTROL

```

```

SUBTRACT:
    MOV   A,PULSE_WIDTH
    SUBB  A,PERIOD
    MOV   PULSE_WIDTH,A
    MOV   A,PULSE_WIDTH+1
    SUBB  A,PERIOD+1
    MOV   PULSE_WIDTH+1,A
    MOV   A,R3
    SUBB  A,#0
    MOV   R3,A

```

```

LOOP_CONTROL:
    DJNZ  R2,TIMES_TWO

```

APPENDIX B Duty Cycle Calculation

Inputs to this routine are 16-bit pulse width and period measurements of a rectangular waveform. The output is a 6-bit BCD number representing the duty cycle of the waveform. The low 8 bits of the result are returned in DUTY_CYCLE. The 8th bit is the carry bit in the PSW. If the duty cycle is between 0 and 99 percent, the carry bit is 0 and DUTY_CYCLE contains the two BCD digits representing the duty cycle as a percent. If the duty cycle is 100 percent, the carry bit is 1 and DUTY_CYCLE contains 0.

INPUTS: PULSE_WIDTH 2 bytes in externally defined DATA
(low byte at PULSE_WIDTH, high byte at PULSE_WIDTH+1)

PERIOD 2 bytes in externally defined DATA
(low byte at PERIOD, high byte at PERIOD+1)

OUTPUT: DUTY_CYCLE 1 byte in externally defined DATA

VARIABLES AND REGISTERS MODIFIED:

PULSE_WIDTH, DUTY_CYCLE

ACC.0, PSW, R2, R3

270609-36

```
MOV A,PULSE_WIDTH
RLC A
MOV PULSE_WIDTH,A
MOV A,PULSE_WIDTH+1
RLC A
MOV PULSE_WIDTH+1,A
MOV A,R3
RLC A
```

FINAL_COMPARE:

```

CJNE R3,#0,FINAL_DONE
MOV A,PULSE_WIDTH+1
CJNE A,PERIOD+1,FINAL_DONE
MOV A,PULSE_WIDTH
CJNE A,PERIOD,FINAL_DONE

```

```

FINAL_DONE:
JC      CONVERT_TO_BCD
MOV     A,DUTY_CYCLE
ADD     A,#1
MOV     DUTY_CYCLE,A
JNC     CONVERT_TO_BCD
CLR     OV
RET

```

```

CONVERT TO BCD:
    MOV     A,DUTY_CYCLE
    MOV     B,#10
    MUL     AB
    XCH     A,B
    SWAP    A
    MOV     DUTY_CYCLE,A
    MOV     A,#10
    MUL     AB
    XCH     A,B
    ORL     DUTY_CYCLE,A
    MOV     A,#10
    MUL     AB
    MOV     A,B
    CJNE    A,#5,TEST
TEST:      JBC     CY,OUT
    MOV     A,DUTY_CYCLE
    ADD     A,#1
    DA      A
    MOV     DUTY_CYCLE,A
OUT:       RET

```

END

270609-37

APPENDIX C

A map of the Special Function Register (SFR) space is shown in Table A1. Those registers which are new or have new bits added for the 83C51FA and 83C51FB have been **boldfaced**.

Note that not all of the addresses are occupied. Unoccupied addresses are not implemented on the chip.

Read accesses to these addresses will in general return random data, and write accesses will have no effect.

User software should not write 1s to these unimplemented locations, since they may be used in future 8051 family products to invoke new features. In that case the reset or inactive values of the new bits will always be 0, and their active values will be 1.

Table A1. Special Function Register Memory Map and Values After Reset

F8		CH 00000000	CCAP0H XXXXXXXX	CCAP1H XXXXXXXX	CCAP2H XXXXXXXX	CCAP3H XXXXXXXX	CCAP4H XXXXXXXX	FF
F0	* B 00000000							F7
E8		CL 00000000	CCAP0L XXXXXXXX	CCAP1L XXXXXXXX	CCAP2L XXXXXXXX	CCAP3L XXXXXXXX	CCAP4L XXXXXXXX	EF
E0	* ACC 00000000							E7
D8	CCON 00X00000	CMOD 00XXX000	CCAPM0 X0000000	CCAPM1 X0000000	CCAPM2 X0000000	CCAPM3 X0000000	CCAPM4 X0000000	DF
D0	* PSW 00000000							D7
C8	T2CON 00000000	T2MOD XXXXXXXX0	RCAP2L 00000000	RCAP2H 00000000	TL2 00000000	TH2 00000000		CF
C0								C7
B8	* IP X0000000	SADEN 00000000						BF
B0	* P3 11111111							B7
A8	* IE 00000000	SADDR 00000000						AF
A0	* P2 11111111							A7
98	* SCON 00000000	* SBUF XXXXXXXX						9F
90	* P1 11111111							97
88	* TCON 00000000	* TMOD 00000000	* TL0 00000000	* TL1 00000000	* TH0 00000000	* TH1 00000000		8F
80	* P0 11111111	* SP 00000111	* DPL 00000000	* DPH 00000000			*PCON ** 00XX0000	87

* = Found in the 8051 core (See 8051 Hardware Description in the Embedded Controller Handbook for explanations of these SFRs).

** = See description of PCON SFR. Bit PCON.4 is not affected by reset.

X = Undefined.

ing the polarity of the voltage applied to the motor. Figure 1 shows a simplified symbolic representation of a driver circuit which is capable of reversing the polarity of the input to the motor.

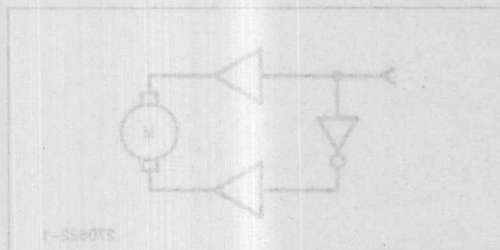


Figure 1. Reversible Motor Driver Circuit

Varying the speed requires changing the voltage level of the input to the motor, and that means changing the input level to the motor driver. In a digitally-controlled system, the analog signal to the driver must come from some form of D/A converter. But adding a D/A converter to the circuit adds to the chip count, which means more cost, higher power consumption, and reduced reliability of the system.

The other alternative is to vary the pulse width of a digital signal input to the motor. By varying the pulse width, the average voltage delivered to the motor changes. A digital circuit can be configured to have up to 2 on-board pulse width modulators.

THE 83C21FA

The 83C21FA is an 8-bit microcontroller based on the 8051 architecture. It is an enhanced version of the 83C01 and incorporates many new features including the Programmable Counter Array (PCA).

Included in the Programmable Counter Array is a 16-bit free running timer and 2 separate modules.

The PCA timer has two 8-bit registers called CL (low byte) and CH (high byte), and is started by all modules. It can be programmed to take input from four different sources. The inputs provide flexibility in choosing the count rate of the timer. The maximum count rate is 4 MHz (1/2 of the oscillator frequency).

Four I/O pins are used to interface each module to the outside world. When the port pins are not used by the PCA modules, they may be used as regular I/O pins.

The modules of the PCA can be programmed to perform in one of the following modes: capture mode,

INTRODUCTION

This application note shows how an 83C21FA can be used to efficiently control DC motors with minimum hardware requirements. It also discusses software implementation and program helpful techniques as well as sample code needed to realize precision control of a motor.

There is also a brief overview of the new features of the 83C21FA. The new feature is called the Programmable Counter Array (PCA) and is capable of delivering Pulse Width Modulated signals (PWM) through designated I/O pins.

September 1988

Order is familiar with the MCS-51 assembly language. For more information about the 8051 architecture and the PCA, refer to the Intel® 8051 Controller Handbook Volume 1 (Order no. 210618-001).

This document will not discuss stepper motors or motor control algorithms.

DC MOTORS

DC motors are widely used in industrial and consumer applications. In many cases, absolute precision in movement is not required, but precise speed control is. For example, a DC motor can be configured to rotate at a constant rate, or to rotate at a certain point.

However, some motor applications do require precise positioning. Examples are high resolution plotters, printers, disk drives, robotics, etc. Stepper motors are frequently used in these applications. There are also applications which require precise speed control along with some position accuracy. Video recorders, compact disk drives, high density recorders are examples of this category.

By controlling DC motors accurately, they can overlap many applications of stepper motors. The cost of the control system depends on the accuracy of the encoder and the speed of the processor.

The 83C21FA can control a DC motor accurately with minimum hardware at a very low cost. The microcontroller can digitally control the speed of a system, can digitally control the position of the motor, by monitoring the output lines and changing the output lines. I/O pins are not used by the PCA modules, they may be used as regular I/O pins.

Almost every application that uses a DC motor requires a means to reverse the direction of rotation or vary its speed. Reversing the direction is simply done by changing the polarity of the input to the motor.

INTRODUCTION

This application note shows how an 83C51FA can be used to efficiently control DC motors with minimum hardware requirements. It also discusses software implementation and presents helpful techniques as well as sample code needed to realize precision control of a motor.

There is also a brief overview of the new features of the 83C51FA. This new feature is called the Programmable Counter Array (PCA) and is capable of delivering Pulse Width Modulated signals (PWM) through designated I/O pins.

It is assumed that the reader is familiar with the MCS-51 architecture and its assembly language. For more information about the 8051 architecture and the PCA refer to the Embedded Controller Handbook Volume 1 (order no. 210918-006).

This document will not discuss stepper motors or motor control algorithms.

DC MOTORS

DC motors are widely used in industrial and consumer applications. In many cases, absolute precision in movement is not an issue, but precise speed control is. For example, a DC motor in a cassette player is expected to run at a constant speed. It does not have to run for precise increments which are fractions of a turn and stop exactly at a certain point.

However, some motor applications do require precise positioning. Examples are high resolution plotters, printers, disk drives, robotics, etc. Stepper motors are frequently used in those applications. There are also applications which require precise speed control along with some position accuracy. Video recorders, compact disk drives, high quality cassette recorders are examples of this category.

By controlling DC motors accurately, they can overlap many applications of stepper motors. The cost of the control system depends on the accuracy of the encoder and the speed of the processor.

The 83C51FA can control a DC motor accurately with minimum hardware at a very low cost. The microcontroller, as the brain of a system, can digitally control the angular velocity of the motor, by monitoring the feedback lines and driving the output lines. In addition it can perform other tasks which may be needed in the application.

Almost every application that uses a DC motor requires it to reverse its direction of rotation or vary its speed. Reversing the direction is simply done by changing

the polarity of the voltage applied to the motor. Figure 1 shows a simplified symbolic representation of a driver circuit which is capable of reversing the polarity of the input to the motor.

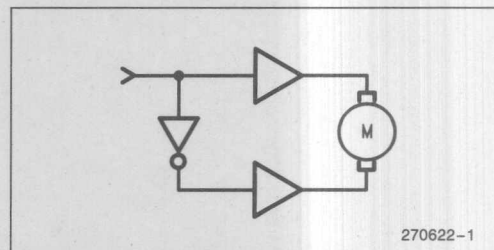


Figure 1. Reversible Motor Driver Circuit

Varying the speed requires changing the voltage level of the input to the motor, and that means changing the input level to the motor driver. In a digitally-controlled system, the analog signal to the driver must come from some form of D/A converter. But adding a D/A converter to the circuit adds to the chip count, which means more cost, higher power consumption, and reduced reliability of the system.

The other alternative is to vary the pulse width of a digital signal input to the motor. By varying the pulse width the average voltage delivered to the motor changes and so does the speed of the motor. A digital circuit that does this is called a Pulse Width Modulator (PWM). The 83C51FA can be configured to have up to 5 on-board pulse width modulators.

THE 83C51FA

The 83C51FA is an 8-bit microcontroller based on the 8051 architecture. It is an enhanced version of the 87C51 and incorporates many new features including the Programmable Counter Array (PCA).

Included in the Programmable Counter Array is a 16-bit free running timer and 5 separate modules.

The PCA timer has two 8-bit registers called CL (low byte) and CH (high byte), and is shared by all modules. It can be programmed to take input from four different sources. The inputs provide flexibility in choosing the count rate of the timer. The maximum count rate is 4 MHz ($\frac{1}{4}$ of the oscillator frequency).

Some of the port 1 pins are used to interface each module and the timer to the outside world. When the port pins are not used by the PCA modules, they may be used as regular I/O pins.

The modules of the PCA can be programmed to perform in one of the following modes: capture mode,

compare mode, high speed output mode, pulse width modulator (PWM) mode, or watchdog timer mode (only module 4).

Every module has an 8-bit mode register called CCAPMn (Figure 2), and a 16-bit compare/capture register called CCAPnL & CCAPnH, where n can be any value from 0 to 4 inclusive. By setting the appropriate bits in the mode register you can program each module to operate in one of the aforementioned modes.

7	6	5	4	3	2	1	0
—	ECOMn	CAPPn	CAPnN	MATn	TOGn	PWMn	ECCCFn

CCAPMn

ECOMn — Enables the comparator function. Must be set for functions which require comparing of the compare/capture registers with the 16-bit timer, i.e., software timer, high-speed output, watchdog timer, and PWM.

CAPPn — Capture on positive edge of signal.

CAPnN — Capture on negative edge of signal.

MATn — Find a match between the capture/compare and 16-bit timer.

TOGn — Toggle I/O pin upon a match between capture/compare registers and 16-bit timer.

PWMn — Generate PWM on I/O pin upon a match between the low byte of capture/compare and the low byte of PCA timer.

ECCCFn — Enables compare/capture flag CCFn in the CCON register to generate an interrupt.

Figure 2. CCAPMn Register

When a module is programmed in capture mode, an external signal on the corresponding port pin will cause a capture of the current value of the 16-bit timer. By setting bits CAPPn or CAPnN or both, the module can be programmed to capture on the rising edge, falling edge, or either edge of the signal. If enabled, an interrupt is generated at the time of capture.

When module is to perform in one of the compare modes (software timer, high speed output, watch dog timer, PWM), the user loads the capture/compare registers with a calculated value, which is compared to the contents of the 16-bit timer, and causes an event as soon as the values match. It can also generate an interrupt.

PWM is one of the compare modes and is the only one which uses only 8 bits of the capture/compare register. The user writes a value (0 to FFH) into the high byte (CCAPnH) of the selected module. This value is transferred into the lower byte of the same module and is compared to the low byte of the PCA timer. While $CL < CCAPnL$ the output on the corresponding pin is a logic 0. When $CL > CCAPnL$, the output is a logic 1.

In this application note we will see how a module can be programmed to perform as a PWM to control the speed and direction of a DC motor.

SETTING UP THE PCA

The 83C51FA has several Special Function Registers (SFRs) that are unknown to ASM51 versions before 2.4. The names of these SFRs must be defined by DATA directive or be defined in a separate file and be included at the time of compilation. Such a file has already been created and is included in the ASM51 package version 2.4.

Two special function registers are dedicated to the PCA timer to allow mode selection and control of the timer. These registers are CCON and CMOD and are shown in figure 3. CCON contains the PCA timer ON/OFF bit (CR), timer rollover flag (CF) and module flags (CCFn). Module flags are used to determine which module causes the PCA interrupt.

7	6	5	4	3	2	1	0
CF	CR	—	CCF4	CCF3	CCF2	CCF1	CCF0

Address 0D8H
Bit Addressable
Reset Value = 00X0 0000 B

CCON

7	6	5	4	3	2	1	0
CIDL	WDTE	—	—	—	CPS1	CPS0	ECF

Address 0D9H
Not Bit Addressable
Reset Value = 00XX X000 B

CMOD

Figure 3. CCON and CMOD Registers

First the clock source for the PCA timer must be defined. The 16 bit timer may have one of four sources for its input. These sources are: osc freq/4, osc freq/12, timer 0 overflow, and external clock.

Two bits in the CMOD register are dedicated to selecting one of the sources for the PCA timer input. They are bits 1 and 2 of CMOD which are called CPS0 and CPS1. CMOD is not bit addressable, thus the value

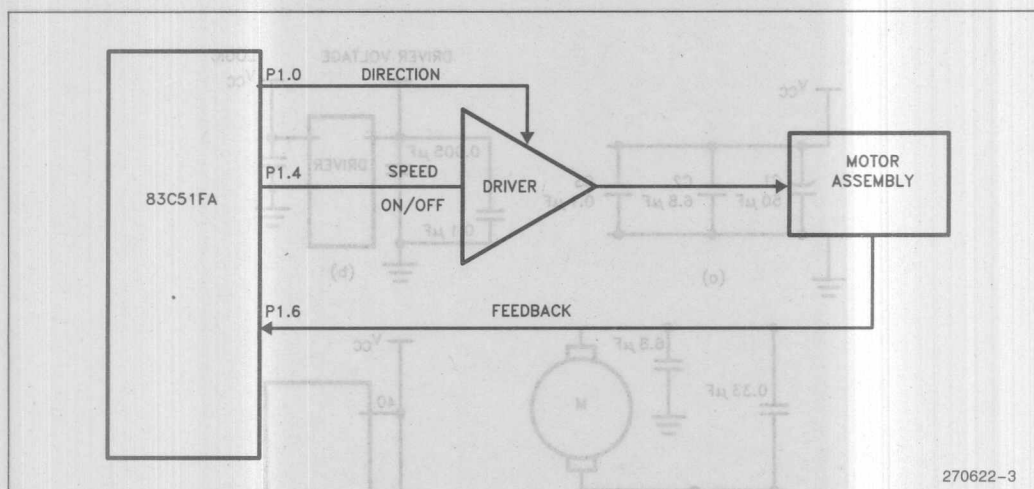


Figure 6. Simplified Circuit Diagram of a Closed Loop System

This configuration, a closed loop circuit, takes up only three I/O pins. The line controlling direction can be a regular port pin but the speed control line must be one of the port 1 pins which corresponds to a PCA module selected for PWM. Depending on how the feedback is generated and processed, it could be connected to a regular I/O, an external interrupt, or a PCA module. Feedback is discussed in more detail in the feedback section of this application note.

The diagram in Appendix A is an example of a DC motor circuit which has been built and bench-tested.

DRIVER CIRCUIT

Although some DC motors operate at 5 volts or less, the 83C51FA can not supply the necessary current to drive a motor directly. The minimum current requirements of any practical motor is higher than any microcontroller can supply. Depending on the size and ratings of the motor, a suitable driver must be selected to take the control signal from the 83C51FA and deliver the necessary voltage and current to the motor.

A motor draws its maximum current when it is fully loaded and starts from a stand still condition. This factor must be taken into account when choosing a driver. However, if the application requires reversing the motor, the current demand will even be higher. As the motor's speed increases, its power consumption decreases. Once the speed of a motor reaches a steady state, the current depends on the load and the voltage across the motor.

Standard motor drivers are available in many current and voltage ratings. One example is the L293 series which can output up to 1 ampere per channel with a supply voltage of 36 V. It has separate logic supply and takes logical input (0 or 1) to enable or disable each channel. There are four channels per device. The L293D also includes clamping diodes needed for protecting the driver against the back EMF generated during the reversing of motor.

NOISE CONSIDERATIONS

Motors generate enough electrical noise to upset the performance of the controller. The source of the noise could be from the switching of the driver circuits or the motor itself. Whatever the cause of the noise may be, it must be isolated or bypassed.

Isolating the microcontroller from the driver circuit is helpful in keeping the noise limited.

Bypass capacitors help a great deal in suppressing the noise. They must be added to the power and ground (Figure 7 diagram a), on the driver circuit (diagram b), on the motor terminals (diagram c), and on the 83C51FA (diagram d). The capacitors must be as close to the component as possible. In fact the best location is under the chip or on top of it if packaging allows. The diagrams in figure 7 show the location and some typical values for the bypass capacitors.

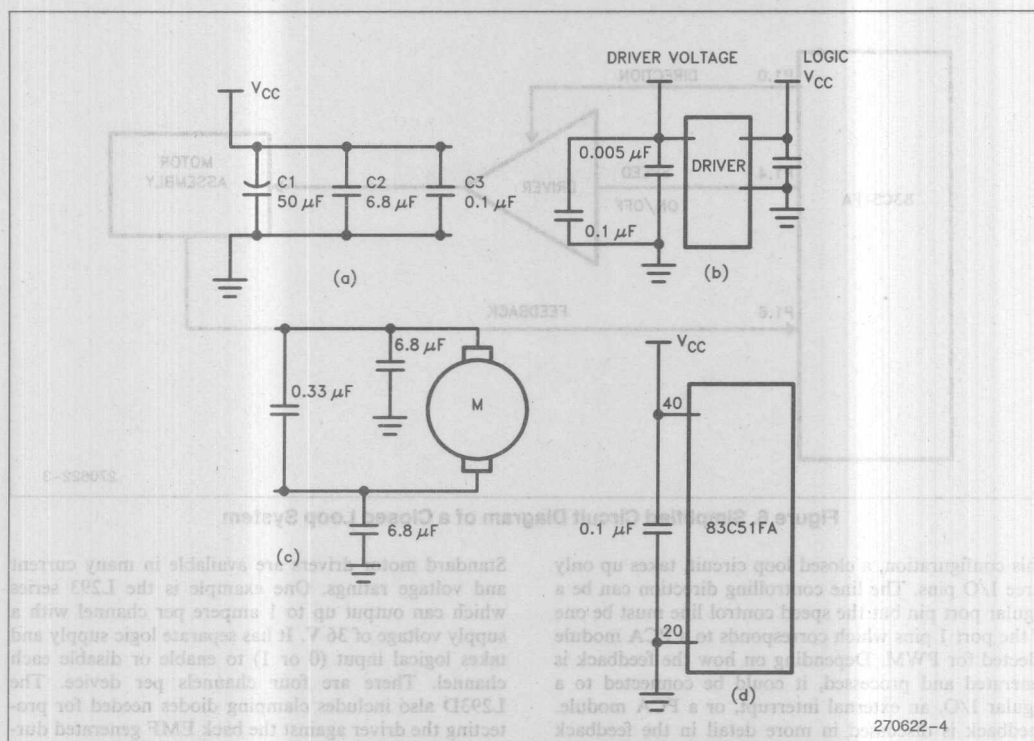


Figure 7. Typical Locations and Values for Bypass Capacitors

OPEN LOOP & CLOSED LOOP SYSTEMS

There are two types of motor control systems: open loop and closed loop.

In the open loop system the controller outputs a signal to turn the motor on/off or to change the direction of the rotation based on an input that does not come from the motor. For example, the position of a manual or timer switch becomes the input to the controller, which varies the input to the motor. In another case, the controller may take input from data tables in the program to run, vary the speed, reverse direction, or stop the motor.

Closed loop systems can use one or more of the above mentioned examples for the open loop system, plus at least one feedback signal from the motor. The feedback signal provides such information as speed, position, and/or direction of motion.

Many applications require that a motor run at a constant speed. The controller has to continuously make adjustments to keep the speed within the limits. In some cases the speed of the motor is synchronized to another motor or moving part of the system.

Depending on the type of feedback signal, the 83C51FA may have to use other modules of the PCA along with other on-chip peripherals such as Timer/Counters, Serial Port, and the interrupt system to precisely control a DC motor.

The example in the following section uses one PCA module to generate PWM, and another module (in capture mode) to receive feedback from a DC motor.

FEEDBACK

The feedback comes from a sensing device which can detect motion. The sensing device may be an optical encoder, infrared detector, Hall effect sensor, etc. Depending on the application, one or more of the above mentioned sensing devices may be suitable.

The optical sensors should be encapsulated for better reliability. If they are not enclosed, factors such as ambient light, dust, and dirt can lessen their sensitivity.

Hall effect sensors are insensitive to any type of light. They change logic levels going into and coming out of a magnetic field. The sensing device is normally mounted

on some stationary part of the system and the magnet is installed on the rotating part. The potential problem with the Hall effect sensors are that if the gap between the magnet and the sensing device is too big, the sensing device may not be affected by the magnetic field. Also the number of magnets is limited which means fewer feedback pulses will be provided.

Whatever the means of sensing, the result is a signal which is fed to the controller. The 83C51FA can use the feedback signal to determine the speed and position of the motor. Then it can make adjustments to increase or decrease the speed, reverse the direction, or stop the motor.

In the following example module 3 of PCA is set up to perform in the capture mode. In this mode module 3 will receive feedback signals from a Hall effect transistor fixed behind a wheel which is mounted on the shaft of a DC motor. Two magnets are embedded in this wheel in equal distances from each other (180 degrees apart). Every time that the Hall effect transistor passes through the magnetic field, it generates a pulse.

The signal is input to P1.6 which is the external interface for module 3 of the PCA. In this example, module 3 is programmed to capture on the rising edge of the

input signal. The time between the two captures corresponds to $\frac{1}{2}$ of a revolution. Thus, two consecutive captures can provide enough information to calculate the speed of the motor as explained in the next paragraph. By storing the value of the capture registers each time, and comparing it to its previous value, the controller can constantly measure and adjust the speed of the motor. Using this method one can run a motor at a precise speed, or synchronize it to another event.

In the PCA interrupt service routine, each capture value is stored in temporary locations to be used in a subtract operation. Subtracting the first capture from the second one will yield a 16-bit result. The resultant value, which will be referred to as "Result" in the rest of this document, is in PCA timer counts. An actual RPM can be calculated from Result. Although the 83C51FA can do the calculation, it would be much faster to provide a lookup table within the code. The table will contain values which have been calculated for a possible range of Results.

The following code is an example of how to measure the period of a signal input to module 3 of the 83C51FA. The diagram in figure 8 shows how the period corresponds to the rotation of the wheel. In the diagram "T" is the period and "t" is the time that the magnet is passing in front of the Hall effect transistor.

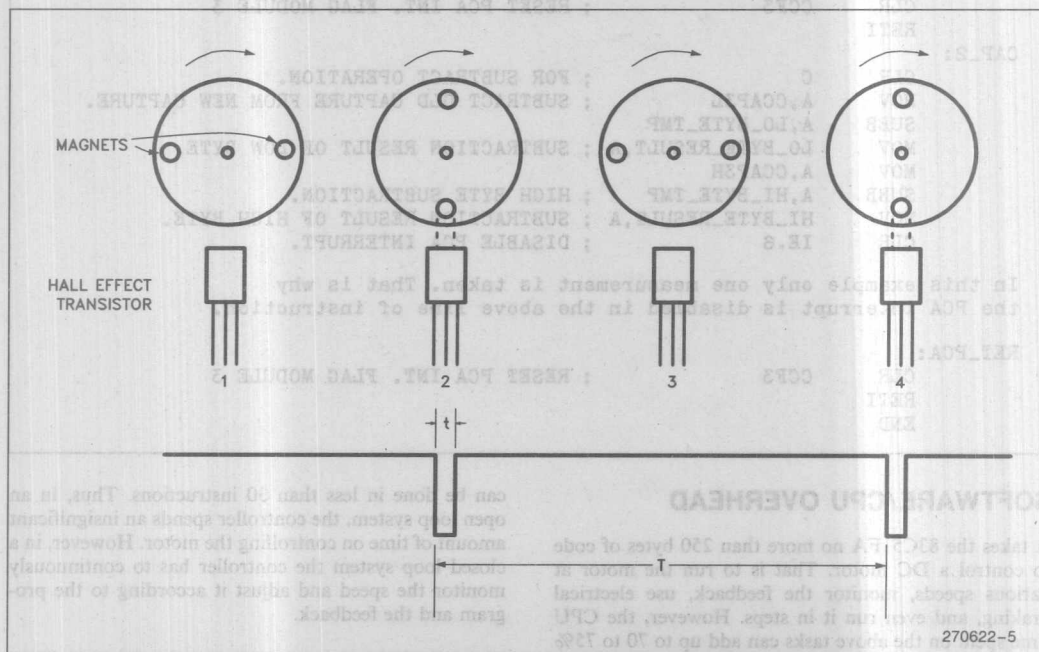


Figure 8. The Output Waveform of the Hall Effect Transistor as it goes Through the Magnetic Field

```

FLAG BIT 0 ; test flag
HI_BYTE_TMP DATA 45H
LO_BYTE_TMP DATA 46H
HI_BYTE_RESULT DATA 47H
LO_BYTE_RESULT DATA 48H

ORG 00H
JMP BEGIN

ORG 33H
JMP PCA_ISR

BEGIN:
MOV CMOD,#0 ; SET PCA TIMER INPUT fOSC/12.
MOV CCAPM3,#21H ; MODULE 3 IN POSITIVE CAPTURE MODE.
MOV CCAP3H,#9AH ; PWM AT 60 PERCENT DUTY CYCLE.
SETB IP.6 ; SET PCA INT. AT HIGH PRIORITY.
MOV IE,#0COH ; ENABLE PCA INTERRUPT.
CLR FLAG ; TURN PCA TIMER ON.
SETB CR

:
:
PCA_ISR:
JB FLAG,CAP_2 ; FLAG BIT IS SET TO SIGNIFY 1st
SETB FLAG ; CAPTURE COMPLETE.
MOV HI_BYTE_TMP,CCAP3H ; SAVE FOR NEXT CALCULATION.
MOV LO_BYTE_TMP,CCAP3L
CLR CCF3 ; RESET PCA INT. FLAG MODULE 3
RETI

CAP_2:
CLR C ; FOR SUBTRACT OPERATION.
MOV A,CCAP3L ; SUBTRACT OLD CAPTURE FROM NEW CAPTURE.
SUBB A,LO_BYTE_TMP
MOV LO_BYTE_RESULT,A ; SUBTRACTION RESULT OF LOW BYTE.
MOV A,CCAP3H
SUBB A,HI_BYTE_TMP ; HIGH BYTE SUBTRACTION.
MOV HI_BYTE_RESULT,A ; SUBTRACTION RESULT OF HIGH BYTE.
CLR IE.6 ; DISABLE PCA INTERRUPT.

In this example only one measurement is taken. That is why
the PCA interrupt is disabled in the above line of instruction.

RET_PCA:
CLR CCF3 ; RESET PCA INT. FLAG MODULE 3
RETI
END

```

SOFTWARE/CPU OVERHEAD

It takes the 83C51FA no more than 250 bytes of code to control a DC motor. That is to run the motor at various speeds, monitor the feedback, use electrical braking, and even run it in steps. However, the CPU time spent on the above tasks can add up to 70 to 75% of the total time available (clock frequency 12 MHz).

The section of software which turns the motor on and off, or sets the speed is very short. In fact, all of that

can be done in less than 30 instructions. Thus, in an open loop system, the controller spends an insignificant amount of time on controlling the motor. However, in a closed loop system the controller has to continuously monitor the speed and adjust it according to the program and the feedback.

The rest of this section talks about electrical braking, stepping a DC motor, and offers examples of code to implement these techniques.

STEPPING A DC MOTOR

Using the 83C51FA, it is possible to run a simple DC motor in small steps. The resolution of the steps will be as high as the resolution of the encoder. If this resolution is sufficient, here is a technique to run a DC motor in steps.

Using a gear box to gear down the motor will increase the resolution of steps. However, putting too much load through the gears will cause sluggish starts and stops.

Electrical braking is used in order to stop the motor at each step. Therefore, the routine that runs the motor in steps will consist of turning it on with full force, waiting for certain period, and stopping it as fast as possible. The wait period depends on the number of steps per revolution.

As the steps and the intervals between them become smaller, the average current demand of the motor increases. This is because the motor is operated at its maximum torque condition every time it starts to rotate and every time it is reversed for electrical braking.

The following code sample shows a continuous loop which runs the motor in steps. The number of steps per revolution depends on the duration of the delay generated by DELAY subroutine. Subroutine WAIT provides the time between the steps.

Subroutine DELAY is the period of time that the motor is kept in reverse. This period must be determined through trial and error for each type of motor and system.

TIME DELAYS

While the 83C51FA is controlling a motor it must frequently wait for the motor to move to certain position before it can proceed with the next task. For example, in the case of electrical braking when the controller reverses the polarity of voltage across the motor, depending on the type, size, and the speed of the motor, it may have up to a second of CPU time before it will turn the motor off.

The wait may be implemented in different ways. Any of the Timer/Counters or unused PCA modules could be utilized to provide accurate timing. The advantage in using the timers is that while the timer is counting, the processor can be taking care of some other tasks. When the timer times out and generates an interrupt the processor will go back and continue servicing the motor.

If there are no timers or PCA modules available for this purpose, a software timer maybe set up by decrementing some of the internal registers. In this method the processor will be tied up counting up or down and will not be able to do anything else. An example of such a timer is:

LOOP:

```
CLR    P1.0          ; SET DIRECTION CLOCKWISE
MOV    CCAP1H,#0      ; MAX PWM
```

The above instruction sets the motor running clockwise. The controller can be doing other tasks if need be, or just stay in a wait loop, then stop the motor as shown below.

```
SETB   P1.0          ; REVERSING THE MOTOR.
MOV     CCAP1H,#0FFH ;
CALL    DELAY         ; WAIT FOR IT TO STOP.
CLR     P1.0          ; REDUCE VOLTAGE TO 0.
CALL    WAIT          ; TIME BEFORE NEXT STEP.
JMP     LOOP
```

```

DELAY:      MOV     R4,#25      ;(decimal)
            MOV     R5,#255    ;(decimal)
DELAY_LOOP: DJNZ    R5,DELAY_LOOP
            DJNZ    R4,DELAY_LOOP
            RET

```

Subroutine DELAY provides approximately 6.4 ms with a 12 MHz clock or 4.8 milliseconds with a 16 MHz clock. The length of this delay can be controlled by loading smaller or larger values to R4 to vary from 260 microseconds up to 65 milliseconds at 12 MHz or 48 milliseconds at 16 MHz oscillator frequency. Larger delays may be obtained by cascading another register and creating an outer loop to this one.

Let us assume that it will take a motor 500 milliseconds to stop from its CW rotation and we are going to use Timer/Counter 0 to provide the wait period. Subroutine DELAY1 will keep track of this timing. Module 1 of PCA is selected to provide the PWM.

```

ORG     OBH
JMP     TIMER_INTERRUPT_ROUTINE
.
CLR     P1.0 ; SET DIRECTION CW
MOV     CCAP1H,#0 ; MAX PWM
Now the motor is running and the controller can do other tasks.
Some typical tasks are called in the following segment.
BUSY_LOOP:
CALL    MONITOR_DISPLAY
CALL    SCAN_KEY_BOARD
CALL    SCAN_INPUT_LINES
JNB     STOP_FLAG,BUSY_LOOP
STOP_FLAG gets set by a feedback signal and denotes that the motor must
stop.
SETB    P1.0 ; REVERSING THE MOTOR
MOV     CCAP1H,#OFFH ;
CALL    DELAY ; WAIT TILL MOTOR STOPS
CLR     P1.0 ; REDUCE VOLTAGE TO 0
.
.
.
DELAY1: SETB    EA
        SETB    ETO ; enable timer 0 interrupt
        MOV     TLO,#0D8H
        MOV     THO,#5EH

```

SETB	TRO	; timer 0 on	DELAY:
MOV	R7,#8	; keep track of how many times	MOV
		; timer 0 must roll over	MOV
			DELAY_LOOP:
			DJNZ
			DJNZ
			RET
MONITOR_LOOP:			
CALL	MONITOR_DISPLAY		
CALL	SCAN_KEY_BOARD		
CALL	SCAN_INPUT_LINES		
JB	TRO, MONITOR_LOOP		
RET			
TIMER_INTERRUPT_ROUTINE:			
DJNZ	R7, FULL_COUNT		
CLR	TRO		
FULL_COUNT			
RET			

To implement a 500 milliseconds delay, timer 0 is used here. In mode 1 timer 0 is a 16-bit timer which takes 65.535 milliseconds at 12 MHz to roll over. Dividing 500 milliseconds to 65.535 shows that timer has to overflow more than 7 times but less than 8 times. How much more than 7 times? The following calculation yields the initial load value of the timer.

$$500 \div 65.535 = 7.6295 \text{ taking it backward}$$

$$65.535 \times 7 = 458.745 \text{ milliseconds}$$

$$500 - 458.745 = 41.255 \text{ milliseconds or } 41255 \text{ microseconds.}$$

In hexadecimal it is A127H. The initial load value is the complement of this value which is 5ED8H.

CONCLUSION

The 83C51FA with all its on-chip peripherals is a system on one chip. It can simplify the design of a control board and reduce the chip count. Up to 5 DC motors can be controlled while doing other tasks such as monitoring feedback lines, human interfacing (scanning a keyboard, displaying information), and communicating with other processors. The MCS-51 powerful instruction set provides maximum flexibility with minimum hardware.

With its onboard program memory capability, the need for the external EPROM and address latch is eliminated. The 83C51FA can have up to 8K bytes of code and the 83C51FB can have up to 16K bytes of code on-board.

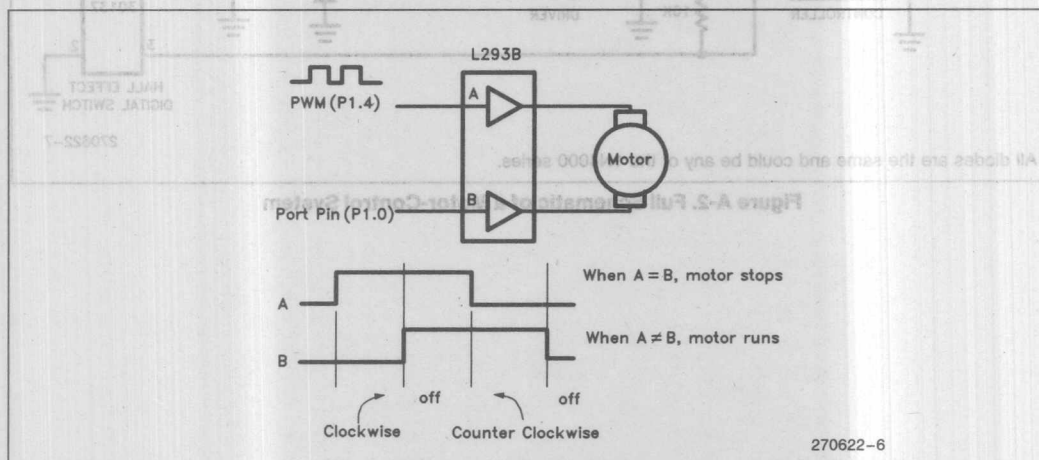
This microcontroller can be used in industrial, commercial, and automotive applications.

APPENDIX A

Figure A-1 shows a symbolic view of the L293B driver. This driver has 4 channels but only two are shown here. Note the inputs A and B and how they are related to each other. You can input the PWM to either one of the inputs and by toggling the other input start or stop the motor. While running, the PWM input controls the

speed. Pin P1.4 corresponds to module 1 of the PCA, and pin P1.0 is used as a regular I/O pin.

Figure A-2 shows the schematic of the motor driver, motor, feedback path, and the supporting components.



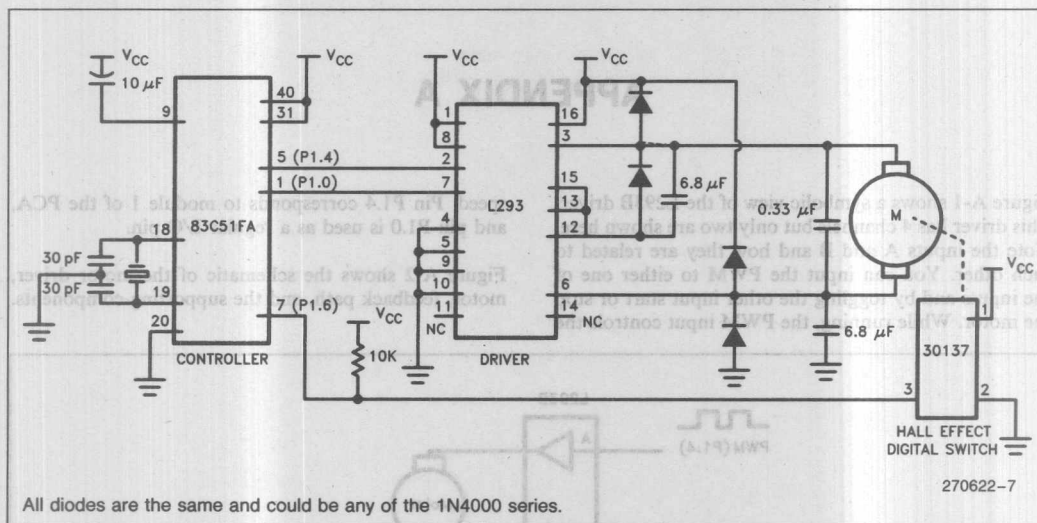


Figure A-2. Full Schematic of a Motor-Control System

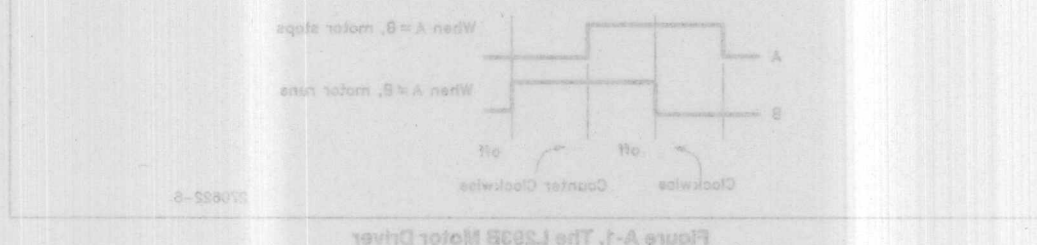


Figure A-1. The L293B Motor Driver

Using the 8051 Microcontroller with Resonant Transducers

TOM WILLIAMSON

Abstract—Having to interface an analog transducer to a digital control system through an analog-to-digital converter represents an expensive bottleneck in the development of many systems. Some transducer companies are addressing this problem by developing proprietary families of resonant transducers.

Resonant transducers are oscillators whose frequency depends in some known way on the physical property being measured. The electrical output from these devices is a train of rectangular pulses whose repetition rate encodes the value of the measurand. Changes in the measurand cause the frequency to shift. The microcontroller detects the frequency shift, runs a validity check on it, and converts it in software to the measurand value.

This paper discusses software interfacing techniques between resonant transducers and the 8051. Techniques for measuring frequency and period are discussed and compared for resolution and interrogation time. The 8051 is capable of performing these tasks in extremely short CPU time. Requirements for obtaining n -bit resolution in the measurement are discussed. It is determined that it is always faster to evaluate the measurand to a given level of resolution by measuring the period rather than the frequency, even if the measurand is proportional to the frequency rather than to the period. Numerical and software examples are presented to illustrate the concepts.

I. RESONANT TRANSDUCERS

MOST sensing transducers are not directly compatible with digital controllers, because they generate analog signals. A few transducer companies are developing proprietary families of sensors which generate signals that are more directly compatible with digital systems. These are not analog sensors with built-in A-D conversion, but oscillators whose frequency depends in some known way on the physical property being measured.

The technology is applicable to virtually any type of measurand: pressure, gas density, position, temperature, force, etc. The sensor and microcontroller can operate from the same supply voltage, so the sensor can in most cases connect directly to a port pin on the microcontroller.

The nominal reference frequency of the output signal from these devices is in the range of 20 Hz–500 kHz, depending on the design. A change in the measurand away from the reference condition causes the frequency to shift by an amount that is related to the change in the measurand value. Transducers are available that have a full-scale frequency shift of 2–1. The microcontroller detects the change in frequency or period and converts it in software to the measurand value.

II. CONNECTING THE DIGITAL TRANSDUCER TO THE 8051

Normally the transducer output can be connected directly to one of the 8051 port pins. An exception would occur when the

transducer signal does not restrict itself to the voltage range of -0.5 to $+5.5$ V.

The 8051 is not sensitive to the rise and fall times of its input signals. It detects transitions by sampling its port pins at fixed intervals (once per machine cycle), and responds to a change in the sequence of samples. If the slew rate of the transducer signal is extremely slow, noise superimposed on the signal could cause the sequence of samples to show false transitions. There could on that account be situations in which the transducer signal should be buffered through a Schmitt Trigger to square it up.

III. TIMER/COUNTER STRUCTURE IN THE 8051

The 8051 has two 16-bit timer/counters: Timer 0 and Timer 1. Both can be configured in software to operate either as timers or as event counters.

In the "timer" function, the register is automatically incremented every machine cycle. Since a machine cycle in the 8051 consists of 12 clock periods, the timer is being incremented at a constant rate of $1/12$ the clock frequency.

In the "counter" function, the register is incremented in response to a 1-to-0 transition at its corresponding external input pin (T0 or T1). The way this function works is the external input pin is sampled once each machine cycle (once every 12 clock periods), and when the samples show a high in one cycle and a low in the next, the count is incremented.

Note too that since it takes two machine cycles (24 clock periods) to recognize a 1-to-0 transition, the maximum count rate is $1/24$ the clock frequency. If the clock frequency is 12 MHz, the maximum count rate is 500 kHz. There are no requirements on the duty cycle of the signal being counted.

The 8052, an enhanced version of the 8051, has three 16-bit timer/counters, two of which are identical to those in the 8051. The third timer/counter can operate either as a 16-bit timer/counter with automatic reload to a preset 16-bit value on rollover, or as a 16-bit timer/counter with a "capture" mode. In the capture mode a 1-to-0 transition at the T2EX pin causes the current value in the counting register to the "captured" into RAM. The third timer makes the 8052 particularly easy to interface with resonant transducers.

IV. WHETHER TO MEASURE FREQUENCY OR PERIOD

Measuring the frequency requires counting transducer pulses for a fixed sample time. Measuring the period requires measuring elapsed time for a fixed number of transducer pulses. For a given level of accuracy in the determination of the value of the measurand, it is usually faster to measure the period, rather than the frequency, even if the measurand is

Manuscript received October 25, 1984.

The author is with Intel Corporation, Chandler, AZ 85224.

WILLIAMSON: USING THE 8051 MICROCONTROLLER

proportional to frequency rather than period. However, both types of measurements will be discussed here.

Two timer/counters can be used, one to mark time and the other to count transducer pulses. If the frequency being counted does not exceed 50 kHz or so, one may equally well connect the transducer signal to an external interrupt pin, and count transducer pulses in software. That frees one timer, with very little cost in CPU time.

V. HOW TO MEASURE TRANSDUCER FREQUENCY

Measuring the frequency means counting transducer pulses for some desired sample time. The count that is directly obtained is $T \times F$, where T is the sample time and F is the frequency. The full scale range is $T \times (F_{\max} - F_{\min})$. For n -bit resolution

$$1 \text{ LSB} = \frac{T \times (F_{\max} - F_{\min})}{2^n}$$

Therefore, the sample time required for n -bit resolution is

$$T = \frac{2^n}{F_{\max} - F_{\min}}$$

For example, 8-bit resolution in the measurement of a frequency that varies between 5 and 10 kHz would require, according to this formula, a sample time of 51.2 ms. The maximum acceptable frequency count would be $51.2 \text{ ms} \times 10 \text{ kHz} = 512$ counts. The minimum would be 256 counts. Subtracting 256 from each frequency count would allow the frequency to be reported on a scale of 0 to FF in hex digits.

If F_{\min} and F_{\max} are closer together it takes more time to resolve them. 8-bit resolution in the measurement of a frequency that varies between 7 and 9 kHz would require a sample time of 128 ms. The maximum and minimum acceptable counts would be 1152 and 896. Subtracting 896 from each frequency count would allow the frequency to be reported on a scale of 0 to FF in hex digits.

To implement the measurement, one timer is used to establish the sample time. In this function it autoincrements every machine cycle. A machine cycle consists of 12 periods of the clock oscillator. The sample time can be converted to machine cycles by multiplying it by $(F_{\text{xtal}})/12$, where F_{xtal} is the 8051 clock frequency. The timer needs to be preset so that it rolls over at the end of each sample time. Then it generates an interrupt, and the interrupt routine reads and clears the transducer pulse counter, and then reloads the timer with the correct preset value.

The preset or reload value is the two's complement negative of the sample time in machine cycles. For example, with a 12-MHz clock frequency, the reload value required to establish a 51.2 ms sample time is

$$-(51.2 \text{ ms}) \times (12000 \text{ kHz}) = -51200 = 3800 \text{ H}$$

In many cases the required sample time exceeds the capacity of a 16-bit timer. For example, establishing a 128 ms sample time with a 12-MHz clock frequency requires a 3-byte timer with a reload of FE0C00H. The 8051 timer, being only 2-

bytes wide, can be augmented in software in the timer interrupt routine to three bytes. The 8051 has a DJNZ instruction (decrement and jump if not zero) which makes it easier to code the third timer byte to count down instead of up. If the third timer byte counts down, its reload value is the two's complement of what it would be for an up-counter. For example, if the two's complement of the sample time is FE0C00H, then the reload value for the third timer byte would be 02, instead of FE. The timer interrupt routine might then be

```
DJNZ THIRD_TIMER_BYTE,OUT
MOV TLO,#0
MOV TH0,#0CH
MOV THIRD_TIMER_BYTE,#02
```

(Now read and clear the transducer pulse counter.)

OUT: RETI

Interrupt latency will have no effect on the measurement if the latency is the same for every sample time.

The trouble with measuring the frequency is it is not only slow, but a waste of the resolving power of the 8051's timers. A timer with microsecond resolution is being used to mark off 100-ms time periods. The technique is nevertheless useful if the timer is already serving other purposes (servicing a display, perhaps), so that the sample time is coming relatively free of charge. But in most cases it is faster and equally accurate to measure the frequency by deriving it from a measurement of the period.

VI. HOW TO MEASURE THE PERIOD

Measuring the period of the transducer signal means measuring the total elapsed time over N -transducer pulses. The quantity that is directly measured is $N \times T$, where T is the period of the transducer signal in machine cycles. The relationship between T in machine cycles and the transducer frequency F in arbitrary frequency units is

$$T = \frac{F_{\text{xtal}}}{F} \times (1/12)$$

where F_{xtal} is the 8051 clock frequency, in the same unit as F .

The full scale range then is $N \times (T_{\max} - T_{\min})$. For n -bit resolution

$$1 \text{ LSB} = \frac{N \times (T_{\max} - T_{\min})}{2^n}$$

Therefore, the number of periods over which the elapsed time should be measured is

$$N = \frac{2^n}{T_{\max} - T_{\min}}$$

However, N must also be an integer. It is logical to evaluate the above formula (do not forget that T_{\max} and T_{\min} have to be in machine cycles) and select for N the next higher integer. This selection gives a period measurement that has somewhat more than n -bit resolution, which may or may not be acceptable, depending on the overall requirements of the

system. It can be scaled back to n -bit resolution, if necessary, by the following computation:

$$\text{reported value} = \frac{NT - NT_{\min}}{NT_{\max} - NT_{\min}}$$

where NT is the elapsed time measured over N periods.

The computation can be done in math if a suitable divide routine is available in the software. For 8-bit resolution it is entirely reasonable to find the reported value in a look-up table, which would take up somewhat more than one page in ROM. In fact, the look-up table would contain $NT_{\max} - NT_{\min}$ entries.

For example, suppose we want 8-bit resolution in the measurement of the period of a signal whose frequency varies from 5 to 10 kHz. If the clock frequency is 12 MHz, then T_{\max} is $(12\,000\text{ kHz})/(12 \times 5\text{ kHz}) = 200$ machine cycles, and T_{\min} is 100 machine cycles. The timer needs to be on then for $N = 2.56$ periods, according to the formula. Using $N = 3$ periods will give maximum and minimum NT values of 600 and 300 machine cycles. This is somewhat more than 8-bit resolution. It can be scaled to 8 bits with a 300-byte look-up table, if desired.

To implement the measurement, one timer is used to measure the elapsed time NT . Enabling its interrupt is optional. The timer interrupt could be used to indicate a short or open in the transducer circuit.

Then the transducer is connected to one of the external interrupt pins (INT0 or INT1), and this interrupt is configured to the transition-activated mode. In the transition-activated mode every 1-to-0 transition in the transducer output will generate an interrupt. The interrupt routine counts transducer pulses, and when it gets to the predetermined N , it reads and clears the timer. For example

```
DJNZ PULSES,OUT
MOV PULSES,N,PERIODS
(Read and clear timer.)
```

```
OUT: RETI
```

If other interrupts are also to be enabled, the one connected to the transducer should be set to Priority 1, and the others to Priority 0. This is to control the interrupt response time. The response time will not affect the measurement if it is the same for every measurement. Variations in the response time will, however, affect the measurement. Setting the pulse-counter interrupt to Priority 1 and all others to Priority 0 will minimize variations in the response time. The response time will then be limited to range from 3 to 8 machine cycles.

VII. PULSEWIDTH MEASUREMENTS

The 8051 timers have an operating mode which is particularly suited to pulsewidth measurements, and may be useful here if the transducer has a fixed duty cycle, or if the transducer output is pulsewidth modulated instead of frequency modulated by the measurand.

In this mode the timer is turned on by the on-chip circuitry in response to an input high at the external interrupt pin, and off by an input low. The external interrupt itself is enabled, so the same 1-to-0 transition from the transducer that turns off the

timer also generates an interrupt. The interrupt routine would then read and reset the timer.

The advantage of this method is that the transducer signal has direct access to the timer gate, with the result that variations in the interrupt response time cease to be a factor. The timer can be read and cleared any time before the next high in the transducer output.

VIII. DERIVING FREQUENCY FROM A PERIOD MEASUREMENT

We now consider the problem of measuring the transducer frequency to n -bit resolution by deriving it from a direct measurement of the period. The advantage of this technique is speed. It is always faster to measure period than frequency. But it is important to end up with a frequency value that has the same resolution and accuracy as a directly measured frequency. Two questions need to be addressed.

- 1) To achieve n -bit resolution in the calculated frequency, how much resolution is required in the period?
- 2) Having measured the period to the required resolution, what is the most efficient way to calculate the frequency?

These questions will be addressed one at a time.

IX. RESOLUTION REQUIREMENTS

In general, n -bit resolution in the frequency derivation requires somewhat more than n -bit resolution in the period measurement. How much more? It will be demonstrated presently that if the transducer frequency varies over a 2-to-1 range, the frequency can be calculated with n -bit resolution from period measurements that have $(n + 1)$ -bit resolution.

The more practical form of the question is over how many periods (N) must the transducer signal be sampled to obtain the required resolution in F ? And so, we commence a calculation of N .

The basic calculation of frequency from $N \times T$ (which we shall call NT) is straightforward:

$$F = N/(NT).$$

The relationship between an increment dF in the calculated frequency due to an increment $d(NT)$ in the measured period is, therefore,

$$dF = -\frac{N}{(NT)^2} d(NT)$$

$$= -\frac{F^2}{N} d(NT).$$

This equation says the value of an LSB in the calculated frequency is $(F^2)/N \times$ the value of an LSB in NT . Then the maximum value that an LSB in the calculated frequency can have is $(F_{\max})^2/N \times$ the value of an LSB in NT . For the calculated frequency to have n -bit resolution over the entire range of frequencies, the value of its LSB must never exceed $(F_{\max} - F_{\min})/2^n$. Therefore, the measurement requires

$$\frac{(F_{\max})^2}{N} \times (1 \text{ LSB in } NT) \leq \frac{F_{\max} - F_{\min}}{2^n}$$

WILLIAMSON: USING THE 8051 MICROCONTROLLER

The required resolution in NT is, therefore,

$$1 \text{ LSB in } NT \leq \frac{N \times (F_{\max} - F_{\min})}{2^n \times (F_{\max})^2}$$

Now, to say that NT is measured with m -bit resolution means

$$1 \text{ LSB in } NT = \frac{N \times (1/F_{\min} - 1/F_{\max})}{2^m}$$

Substituting this value for 1 LSB into the required resolution and solving for 2^m yields

$$2^m \geq \frac{F_{\max}}{F_{\min}} \times 2^n$$

Then the requirement on m is

$$m \geq n + \ln(F_{\max}/F_{\min}) / \ln(2)$$

It can be stated with some certainty, then, that if the transducer frequency varies over a range of 2-to-1, the frequency can be calculated with 8-bit resolution from a period measurement that has 9-bit resolution. If the frequency variation is less than 2-to-1, another full bit of resolution in the period measurement is not needed.

To obtain m -bit resolution in NT , N must satisfy

$$N \geq \frac{2^m}{T_{\max} - T_{\min}}$$

Substituting for 2^m , and using $T_{\max} = 1/F_{\min}$ and $T_{\min} = 1/F_{\max}$, gives the result that

$$N \geq \frac{(F_{\max})^2}{F_{\max} - F_{\min}} \times 2^n$$

It should be noted that the units of frequency here are periods/machine cycle, since the 8051 measures time by counting machine cycles. The conversion factor between Hz and periods/machine cycle is $12/(\text{clock frequency})$. So the requirement on N can also be written

$$N \geq \frac{F_{\max}}{F_{\max} - F_{\min}} \times \frac{F_{\max}}{F_{\text{xtal}}} \times 12 \times 2^n$$

where F_{xtal} is the 8051 clock frequency in the same units as F_{\max} and F_{\min} . This is the number of transducer pulses over which the transducer signal must be sampled to achieve the required resolution in F .

For example, suppose that 8-bit resolution is required in F , where $F_{\max} = 10 \text{ kHz}$ and $F_{\min} = 5 \text{ kHz}$, and that $F_{\text{xtal}} = 12 \text{ MHz}$. Then the above calculation shows that $N = 6$ periods gives sufficient resolution in the period measurement to satisfy the resolution requirement in F . Six periods will take 0.6–1.2 ms of sampling time, on that frequency range. Recall that the sample time for a direct frequency measurement of the same signal and to the same resolution was earlier calculated to be 51.2 ms.

X. COMPUTING THE FREQUENCY FROM THE PERIOD

The period measurement leaves one with a 16-bit integer, which is $N \times T$ (or NT) in machine cycles. The conversion to frequency is straightforward:

$$F = N/(NT) \text{ periods/machine cycle.}$$

The quantity of interest is probably not F , but a normalized measure of the amount by which F exceeds its minimum acceptable value. This quantity represents, through the transducer's transfer function, the "reported value" of the measurand, and this quantity is an n -bit integer whose value ranges from 0 (all bits = 0) to full scale (all bits = 1). This normalized frequency is

$$f = \frac{F - F_{\min}}{F_{\max} - F_{\min}} = \frac{F_{\min}}{F_{\max} - F_{\min}} \times (F/F_{\min} - 1)$$

Using $F = N/(NT)$ and $F_{\min} = N/(NT_{\max})$ allows the normalized frequency to be written

$$f = \frac{F_{\min}}{F_{\max} - F_{\min}} \times \frac{NT_{\max} - NT}{NT}$$

To get a handle on what kinds of numbers are involved here, consider the situation where 8-bit resolution is required in f , and in which $F_{\text{xtal}} = 12 \text{ MHz}$, $F_{\max} = 10 \text{ kHz}$, and $F_{\min} = 5 \text{ kHz}$. We have previously determined that for this set of conditions, $N = 6$ periods gives sufficient resolution in the period measurement to satisfy the resolution requirement in F (and in f). With a 12 MHz clock frequency, T_{\max} in machine cycles is $(12\,000 \text{ kHz})/(12 \times 5 \text{ kHz}) = 200$ machine cycles, so NT_{\max} is $6 \times 200 = 1200$ machine cycles. The calculation for f then becomes

$$f = \frac{1200 - NT}{NT}$$

The minimum acceptable value that NT can have is $(N \times T_{\min} + 1)$, where $T_{\min} = (12\,000 \text{ kHz})/(12 \times 10 \text{ kHz}) = 100$ machine cycles. Then $N \times T_{\min} = 6 \times 100 = 600$ machine cycles. The allowable values for NT are then 601–1200 machine cycles, a total of 599 different values.

The fastest way to "calculate" f would be with a 599-byte look-up table. This method has the added advantage that nonlinearities in the transfer function between frequency and measurand can be built into the table. Look-up tables are facilitated in the 8051 by the `MOVC A, @A + PC`, and `MOVC A, @A + DPTR` instructions. `DPTR` is a 16-bit "data pointer" register in the 8051. Its two bytes can be individually addressed as `DPL` (low byte) and `DPH` (high byte).

In the example under discussion, it will be necessary to load `DPTR` with the address of the first byte of the look-up table, less 601, plus the 2-byte value of NT . The software that accesses the table might then take the following form:

TABLE EQU (address of first table entry)

```

DIVIDE ROUTINE
THIS DIVIDE ROUTINE CALCULATES
    numerator
    denominator
    quotient
    remainder
    in which numerator and denominator are integers and
    numerator / denominator quotient is of the form
    quotient = Q1 Q2 Q3 ... QN
    where Q1 is the coefficient of 2^(n-1). The procedure is

```

Fig. 1. A divide algorithm.

```

MOV A,#LOW(TABLE-601)
ADD A,NTLO
MOV DPL,A
MOV A,#HIGH(TABLE-601)
ADDC A,NTLHI
MOV DPH,A
CLR A
MOVC A,@A+DPTR

```

At this point the accumulator contains the 8-bit value of f .

It is perfectly reasonable to decide that a 599-byte look-up table is unwieldy. Its advantages are speed and built-in error correction. But a reasonably fast divide algorithm can be written to this specific purpose, making use of *a priori* knowledge about the sizes of the numbers that are involved in the computation. It helps to know that in this example the numerator is never going to be larger than 599 and the denominator is always greater than the numerator.

A complete discussion of divide routines is beyond the scope of this paper, but a suitable divide algorithm for this specific application is shown in Fig. 1. Reference [1] calls this the Restoring division algorithm. It is particularly well suited to the 8051, because "<" comparisons are greatly facilitated by the 8051's CJNE (compare and jump if not equal) instruction. CJNE A,B,rel, executes a relative jump if A does not equal B. More importantly to this application, the instruction sets the carry flag if $A < B$.

XI. ACCURACY AND RESOLUTION

The accuracy with which the 8051 will measure the frequency or period of the transducer signal depends on two things: the accuracy of the clock oscillator and variations in the interrupt response time.

Since the clock signal is normally generated by a crystal oscillator, the oscillator accuracy normally far exceeds the quantizing error inherent in the finite (n -bit) resolution.

As was previously mentioned, interrupt response time does not introduce an error into the measurement itself, but variations in the interrupt response time can. Interrupt response time in the 8051 can vary from 3 to 8 machine cycles, depending on what instruction is in progress at the time the interrupt is generated. This would represent an error of ± 5 counts in the measured value of NT during a period measurement. An error of ± 5 counts in NT does not necessarily translate to ± 5 LSB's in the final result, but it might still represent an error that exceeds the resolution.

In a direct frequency measurement variations in the interrupt response time would represent an error of $\pm 5 \mu s$ in the sample time.

If these kinds of errors are unacceptable there are ways to deal with them. In period measurements, if the duty cycle of the transducer is constant, the pulsewidth measurement technique, previously described, can be used. Its advantage is that it gates the timer off when the interrupt is generated, rather than when the interrupt is responded to.

In other cases one can simply increase the sample time above the minimum required to obtain the desired resolution. For example, if the measurement requires 8-bit resolution, one can design the software for an 11-bit resolution and truncate the result to 8 bits.

REFERENCES

- [1] Davio *et al.*, *Digital Systems with Algorithm Implementation*. New York: Wiley, 1983.

ANALOG/DIGITAL PROCESSING WITH MICROCONTROLLERS

John Katausky, Ira Horden, Lionel Smith
Application Engineers
Intel Corp.
5000 W. Williams Field Road
Chandler, AZ 85224

Microcontrollers are rapidly becoming the backbone of silicon computing systems. From a technical standpoint, the most significant attribute, aside from the inclusion of RAM and ROM, that segregates a microcontroller from a microprocessor is I/O manipulation. In general, I/O manipulation is an intimate part of a microcontroller's architecture. The instruction set and architecture of a microcontroller allows the CPU to directly control the I/O facilities on the device. This is in direct contrast to a microprocessor where the I/O is essentially a "sea" of addresses and it is up to the hardware designer to place some type of I/O hardware in this I/O "sea". It should be obvious that simply adding ROM and RAM to a microprocessor WILL NOT create a microcontroller.

This intimate contact with I/O gives the microcontroller a distinct advantage over the microprocessor in applications that are I/O intensive. Microcontrollers can test, set, complement, or clear I/O port pins much faster than a microprocessor and they can also make decisions, based on the state of other hardware features, such as timer/counters with equal speed. This integration of I/O, in both hardware and software makes the microcontroller "ideal" for many types of intelligent instrumentation.

4K ROM/EPROM - 8K ROM ON 8052
128 BYTES OF RAM - 256 ON THE 8052
2-16 BIT TIMER/COUNTERS - 3 ON THE 8052
FULL DUPLEX UART
5 VECTORED INTERRUPTS - 6 ON THE 8052
4 REGISTER BANKS
BIT MANIPULATION (BOOLEAN PROCESSOR)
32 DIRECTLY ADDRESSABLE I/O PINS
MULTIPLY AND DIVIDE INSTRUCTIONS
SUPPORTS 64K OR RAM AND ROM-128K TOTAL

TABLE 1. A BRIEF LISTING OF THE MCS-51'S FEATURE SET.

Intel's MCS-51 series of microcontrollers contain many features that can be integrated directly into many types of instruments. TABLE 1 is a brief listing of these features. To illustrate the power of the 8051 this paper will elaborate on two techniques for performing analog to digital (A to D) conversion. Both of these examples assume that some additional hardware is attached to the I/O pins of the 8051.

S/A CONVERSION TECHNIQUES

Successive approximation analog to digital conversion involves a "binary search" of an unknown voltage relative to a "fixed" known reference. The reference is selectively divided by multiples of two until the desired accuracy is reached. Figure 1 is a flowchart of a successive approximation converter. This technique usually requires a digital to analog converter to divide the reference voltage and a voltage comparator to compare the unknown voltage to the "divided" reference. Digital to analog converters and voltage comparators are readily available and relatively inexpensive. A block diagram of an 8051 based A to D converter is shown in Figure 2.

Many industrial A to D converters require 12 bits of accuracy. A 12 bit converter provides good "dynamic range" and is capable of resolving 1 part in 4096. If the applied input voltage ranges from 0 to 10 Volts, a 12 bit converter can resolve 2.4 millivolts within this range. The theoretical accuracy of a 12 bit converter is .024% +/- 1/2 least significant bit.

The power of the 8051 in this type of application is best revealed by examining the software required to implement the successive approximation algorithm. The routine for the 8051 is shown in Table 2.

The execution times given assume a 12 Mhz crystal. Compare this to the following routine which is a 4 Mhz Z-80

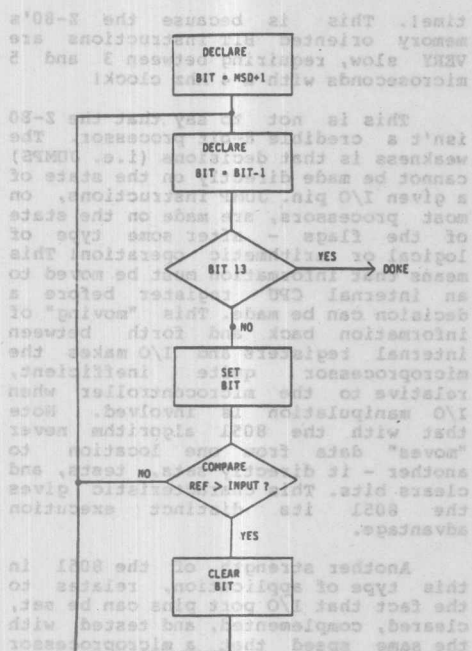


FIGURE 1. SUCCESSIVE APPROXIMATION
CONVERSION ALGORITHM

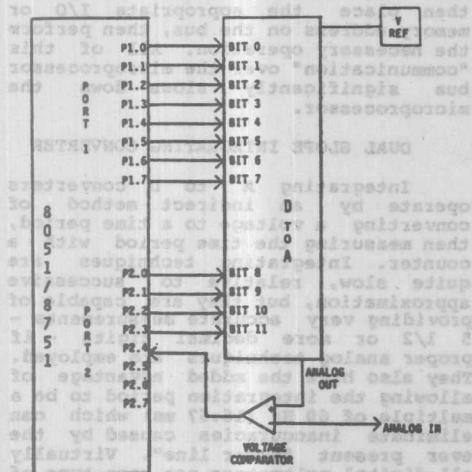


FIGURE 2. BLOCK DIAGRAM OF SUCCESSIVE APPROXIMATION A/D CONVERTER

executing the same algorithm with the 8051. This is because the 8051's memory organization and instructions are very slow, requiring several microseconds with the 8051.

TABLE 2. SUCCESSIVE APPROXIMATION
ROUTINE FOR THE 8051.

	INSTRUCTION	BYTES	TIME
	; CLEAR PORT PINS		
	MOV P1, #0	3	2
	ANL P2, #0F0H	3	2
	; START CONVERSION		
	SETB P2.3	2	1
	JNB P2.4, L1	3	2
	CLR P2.3	2	1
L1:	SETB P2.2	2	1
	JNB P2.4, L2	3	2
	CLR P2.2	2	1
L2:	SETB P2.1	2	1
	JNB P2.4, L3	3	2
	CLR P2.1	2	1
L3:	SETB P2.0	2	1
	JNB P2.4, L4	3	2
	CLR P2.0	2	1
L4:	SETB P1.7	2	1
	JNB P2.4, L5	3	2
	CLR P1.7	2	1
L5:	SETB P1.6	2	1
	JNB P2.4, L6	3	2
	CLR P1.6	2	1
L6:	SETB P1.5	2	1
	JNB P2.4, L7	3	2
	CLR P1.5	2	1
L7:	SETB P1.4	2	1
	JNB P2.4, L8	3	2
	CLR P1.4	2	1
L8:	SETB P1.3	2	1
	JNB P2.4, L9	3	2
	CLR P1.3	2	1
L9:	SETB P1.2	2	1
	JNB P2.4, L10	3	2
	CLR P1.2	2	1
L10:	SETB P1.1	2	1
	JNB P2.4, L11	3	2
	CLR P1.1	2	1
L11:	SETB P1.0	2	1
	JNB P2.4, L12	3	2
	CLR P1.0	2	1
	; CONVERSION COMPLETE		

TOTAL 90 46 US

NOTE: TIMING IS TYPICAL
WORST CASE = 52 US
BEST CASE = 40 US

executing the same algorithm with the D to A hardware attached to an I/O port is shown in Table 3 (assume that all bits on PORT3 are grounded, except the comparator input).

TABLE 3. SUCCESSIVE APPROXIMATION ROUTINE FOR THE Z-80.

	INSTRUCTION	BYTES	TIME
	; CLEAR PORT PINS		
	LD A,0	2	1.75
	OUT (PORT1),A	2	2.75
	OUT (PORT2),A	2	2.75
	; START CONVERSION		
	LD A,08H	2	1.75
	OUT (PORT2),A	2	2.75
	IN A,(PORT3)	2	2.75
	OR A	1	1.00
	IN A,(PORT2)	2	2.75
	JP Z,L1	3	2.50
	AND 0F7H	2	1.75
L1:	OR 04H	2	1.75
	OUT (PORT2),A	2	2.75
	IN A,(PORT3)	2	2.75
	OR A	1	1.00
	IN A,(PORT2)	2	2.75
	JP Z,L2	3	2.50
	AND 0FBH	2	1.75
L2:	OR 02H	2	1.75

REPEAT BETWEEN L1 AND L2 10
MORE TIMES AND SET/RESET THE
APPROPRIATE I/O BITS

TOTAL 179 180 US

AGAIN TIMING IS TYPICAL

WORST CASE = 190.25 US

BEST CASE = 169.25 US

One may argue that by "memory mapping" the Z-80's I/O ports the execution time could be enhanced because the user could take advantage of the Z-80's SET and RESET memory BIT instructions. In reality, a few bytes of memory are saved, but very little

time! This is because the Z-80's memory oriented BIT instructions are VERY slow, requiring between 3 and 5 microseconds with a 4 Mhz clock!

This is not to say that the Z-80 isn't a credible 8-bit processor. The weakness is that decisions (i.e. JUMPS) cannot be made directly on the state of a given I/O pin. JUMP instructions, on most processors, are made on the state of the flags - after some type of logical or arithmetic operation! This means that information must be moved to an internal CPU register before a decision can be made. This "moving" of information back and forth between internal registers and I/O makes the microprocessor quite inefficient, relative to the microcontroller when I/O manipulation is involved. Note that with the 8051 algorithm never "moves" data from one location to another - it directly sets, tests, and clears bits. This characteristic gives the 8051 its distinct execution advantage.

Another strength of the 8051 in this type of application, relates to the fact that I/O port pins can be set, cleared, complemented, and tested with the same speed that a microprocessor can act on its internal registers. Note that the 8051 takes only 1 microsecond to fetch an opcode and set or clear a port pin. A microprocessor must first fetch and decode the opcode, then place the appropriate I/O or memory address on the bus, then perform the necessary operation. All of this "communication" over the microprocessor bus significantly slows down the microprocessor.

DUAL SLOPE INTEGRATING CONVERTER

Integrating A to D converters operate by an indirect method of converting a voltage to a time period, then measuring the time period with a counter. Integrating techniques are quite slow, relative to successive approximation, but they are capable of providing very accurate measurements - 5 1/2 or more decimal digits - if proper analog techniques are employed. They also have the added advantage of allowing the integration period to be a multiple of 60 Hz (16.67 ms) which can eliminate inaccuracies caused by the ever present "power line". Virtually all digital voltmeters use some type of integrating technique. Figure 3 is a block diagram of a typical integrating

A to D converter.

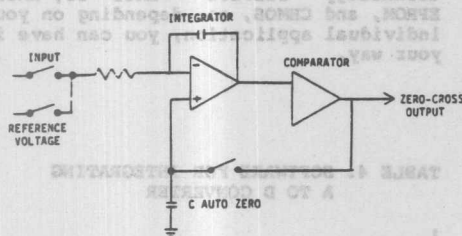


FIGURE 3. INTEGRATING A TO D CONVERTER

Figures 4A, 4B, and 4C show the three typical phases involved in the dual slope technique. Figure 4A illustrates the auto-zero phase. In this phase the integrating "loop" is closed and the offset of the analog integrator is accumulated in C auto zero. In Figure 4B, the input switch is closed and the integrator integrates the input voltage for a fixed time period T₁. In figure 4C, the reference switch is closed and the integrator integrates the reference voltage until the comparator senses a zero crossing condition. The time it takes for this phase to occur is directly proportional to the amplitude of the input voltage. Additional circuitry can be added to determine the polarity of the input voltage, then switch in a reference of opposite polarity, but the basic technique remains the same.

The 8051 is an ideal controller for an intelligent integrating A to D system. The 16 bit timer/counters can provide better than 1/2 decimal digits of accuracy, the serial port can be used to transmit the analog reading to a printer or another processor, the CPU can be interrupted by the 60 Hz line so conversions can start at precise intervals, and software can be used to calculate and save average, peak, or RMS readings.

Another "nice" benefit of this type of converter is that very few I/O port pins are required to control the A to D hardware, so opto-isolators can be used to completely isolate the 8051

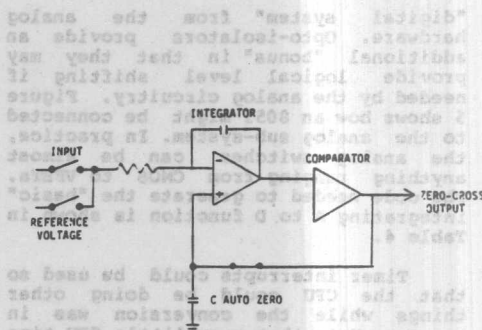


FIGURE 4A. AUTO ZERO PHASE

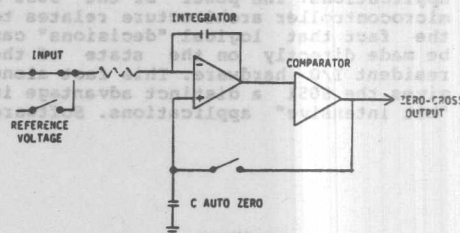


FIGURE 4B. INPUT INTEGRATION PHASE

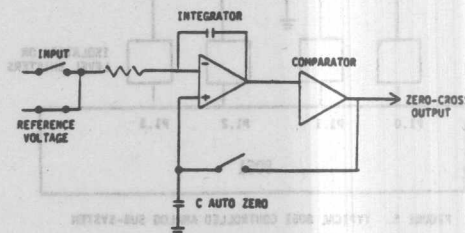


FIGURE 4C. REFERENCE INTEGRATION PHASE

ASIC Family Application Note & Article Reprint

3

While the MCS-21 family is one of the most popular microcontrollers ever introduced, this shared functionality hinders its use in many applications. For example, a "fully loaded" MCS-21-based design would generally leave only one 8-bit port (Port 1) for the application's I/O requirements.

Using Intel's AS² Expand the Capabilities of 80C51-Based Systems

CPO TECHNICAL MA

Cell Name	ROM	RAM
UCB2100	16K ROM	256 Bytes RAM
UCB208	8K ROM	256 Bytes RAM
UCB204	4K ROM	256 Bytes RAM
UCB200	No ROM	256 Bytes RAM
UCB196	16K ROM	128 Bytes RAM
UCB192	8K ROM	128 Bytes RAM
UCB188	4K ROM	128 Bytes RAM
UCB184	No ROM	128 Bytes RAM
UCB180	16K ROM	64 Bytes RAM
UCB176	8K ROM	64 Bytes RAM
UCB172	4K ROM	64 Bytes RAM
UCB168	No ROM	64 Bytes RAM

C Core Cell to Capabilities of an ed System

80C51-BASED ASIC CORE

INSEND MARKETING MANAGER

Using Intel's ASIC Core Cell to Expand the Capabilities of an 80C51-Based System

MATT TOWNSEND
CPO TECHNICAL MARKETING MANAGER

INTRODUCTION

Intel's new ASIC family of microcontroller core cells extends the capability of the MCS[®]-51 product, and allows the ASIC designer more flexibility than the popular microcontroller product. This note will discuss many of the new design possibilities inherent to the 80C51 cell-based controller. This family of cells is available with a variety of RAM and ROM configurations.

Cell Name	ROM	RAM
UC5100	No ROM	128 Bytes RAM
UC5104	4K ROM	128 Bytes RAM
UC5108	8K ROM	128 Bytes RAM
UC5116	16K ROM	128 Bytes RAM
UC5200	No ROM	256 Bytes RAM
UC5204	4K ROM	256 Bytes RAM
UC5208	8K ROM	256 Bytes RAM
UC5216	16K ROM	256 Bytes RAM

Other documentation will address Intel's ASIC design environment (see reference section).

The 80C51-based ASIC cell is part of a family of cell-based functions based on popular Intel standard products. Members of the 82Cxx microprocessor support peripheral family (SP8254, SP8237, SP8259, SP8284, SP82284, SP8288 and SP82288) are also available as library elements. The standard product ASIC cores are supported by a library of over 150 logic cells, representing a broad range of SSI, MSI, and I/O functions. Another class of cell library elements is designated Special Functions. These cells are predefined complex functions such as RAM, Serial I/O, A/D Converter, and a Voltage Comparator. The Special Function and general logic element cells can also be used without a standard product core in the ASIC design. Any of the available 80C51-based cores can be integrated with logic complexities up to 5000 gates.

80C51-BASED ASIC CORE

Although the 80C51-based core is functionally identical to the standard 80C51BH microcontroller, its use as a cell in the ASIC library allows more flexibility in system design and partitioning.

Figure 1 depicts the difference between the standard pinout of the MCS-51 family and the ASIC core. In order to understand the enhancements (in an applications sense) made to the core it is useful to compare its connections to the pinout of the standard product.

The MCS-51 family embodies a very powerful architecture. While it was intended as a "single chip solution"

its addressing modes, clean bus interface, on-chip peripherals, and code efficient instruction set operations make it well suited to processor-like applications as well. For processor applications, a designer forgoes many of the "single chip" features in favor of the high performance CPU functions of this architecture.

In order to fit the MCS-51 family microcontrollers into an economical forty lead DIP or forty-four lead PLCC package, Intel designed the standard product with many of the device's functions sharing pins. The microcontroller designer must compare necessary functions against the economics and performance required for a given design. If external memory or memory mapped I/O is required, then the use of the port 0 function is not available. If the memory address is beyond the 256 byte boundary defined by the AD0-7 Bus then all or part of the port 2 function is not available. Likewise, using peripheral functions like the counter input pins, serial I/O, and interrupts eliminates port 3 functions. While the MCS-51 family is one of the most popular microcontrollers ever introduced, this shared functionality hinders its use in many applications. For example, a "fully loaded" MCS-51-based design would generally leave only one 8-bit port (Port 1) for the application's I/O requirements.

The standard cell version of the 80C51 provides the designer with 116 signals for connection to application specific logic. These signals represent the full function set of the MCS-51 architecture and virtually eliminate any design trade-offs required to implement an application. Notice from Figure 1 that all of the I/O ports are separated from the other functions. In the design example, the I/O are separated into their respective inputs and outputs, leaving 32 inputs and 32 outputs for port connections into the application's logic. The most immediate impact of demultiplexing the I/O of the device is that much of the logic required to complete an application is eliminated. For example, when separating the address from the data on the AD-bus, an octal latch is required. For an 80C51-based core application, the designer uses the A0-7 bus directly, thus saving approximately 100 gates. The fact that the 80C51-based core has so many connections available does not mean an application will be forced into higher pin count packages. A 80C51-based ASIC can implement many system functions more economically than a discrete implementation. The design example illustrates a system with over 280 interconnects that can be integrated into one ASIC device. This application note will illustrate the less obvious ways in which the core can be used.

The illustrations shown in this note are independent of the workstation platform used to implement the design.

Intel provides the complete test vectors necessary to test the 80C51-based ASIC core, which have been derived from the standard product 80C51 test vector set.

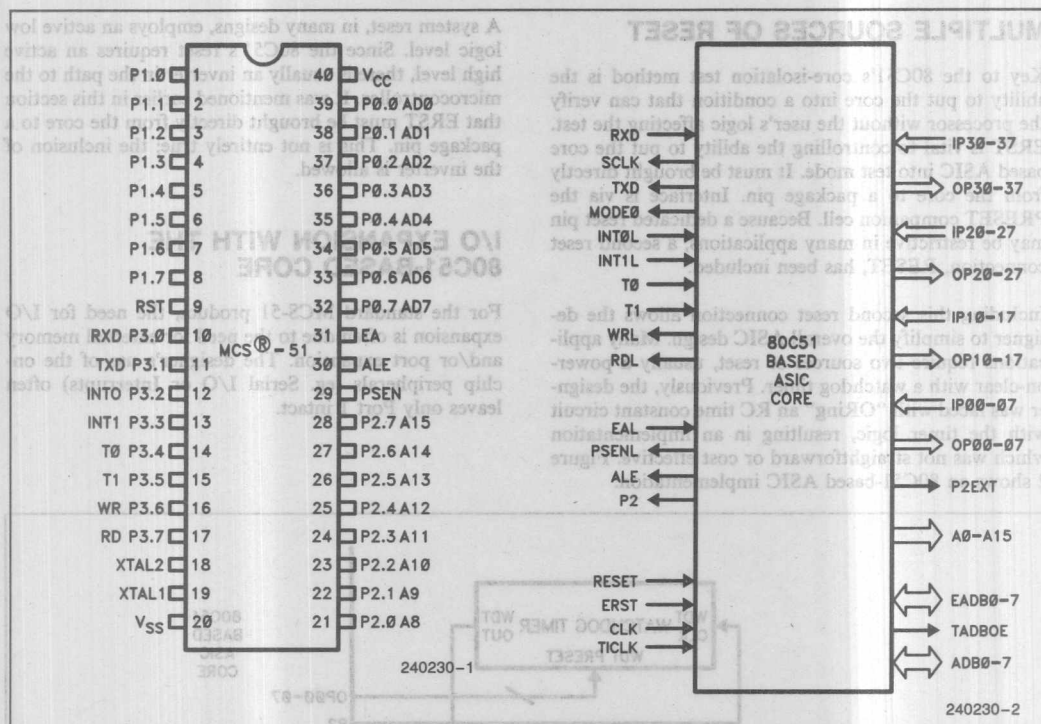


Figure 1. Comparing 80C51 Pin Assignments to Core Connections

RECONSTRUCTION OF STANDARD PRODUCT I/O

When designing the 80C51-based ASIC core, Intel removed the pin multiplexers and I/O functions of the 80C51, and restructured them as companion cells. Companion cells allow the ASIC designer to reconfig-

ure the ASIC cell to function exactly like the standard product. Alternatively, the designer can choose to reconstruct a subset of the standard product I/O or select no reconstruction at all. Consult the references for more information about the use and function of Intel's companion cells.

MULTIPLE SOURCES OF RESET

Key to the 80C51's core-isolation test method is the ability to put the core into a condition that can verify the processor without the user's logic affecting the test. ERST is vital to controlling the ability to put the core based ASIC into test mode. It must be brought directly from the core to a package pin. Interface is via the PRESET companion cell. Because a dedicated reset pin may be restrictive in many applications, a second reset connection, RESET, has been included.

Including this second reset connection allows the designer to simplify the overall ASIC design. Many applications require two sources of reset, usually a power-on-clear with a watchdog timer. Previously, the designer was faced with "ORing" an RC time constant circuit with the timer logic, resulting in an implementation which was not straightforward or cost effective. Figure 2 shows an 80C51-based ASIC implementation.

A system reset, in many designs, employs an active low logic level. Since the 80C51's reset requires an active high level, there is usually an inverter in the path to the microcontroller. It was mentioned earlier in this section that ERST must be brought directly from the core to a package pin. This is not entirely true; the inclusion of the inverter is allowed.

I/O EXPANSION WITH THE 80C51-BASED CORE

For the standard MCS-51 product, the need for I/O expansion is often due to the need for external memory and/or port expansion. The designer's use of the on-chip peripherals (eg. Serial I/O or Interrupts) often leaves only Port 1 intact.

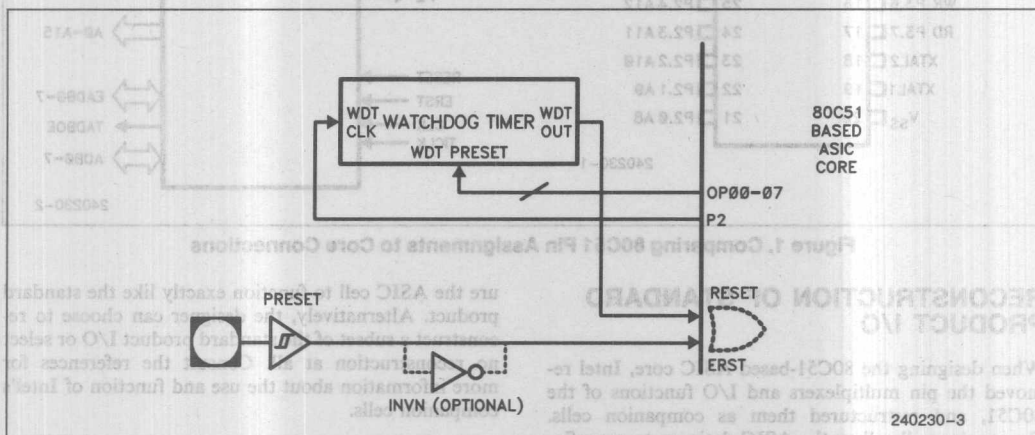


Figure 2. Multiple Sources of Reset for 80C51-Based ASIC Core

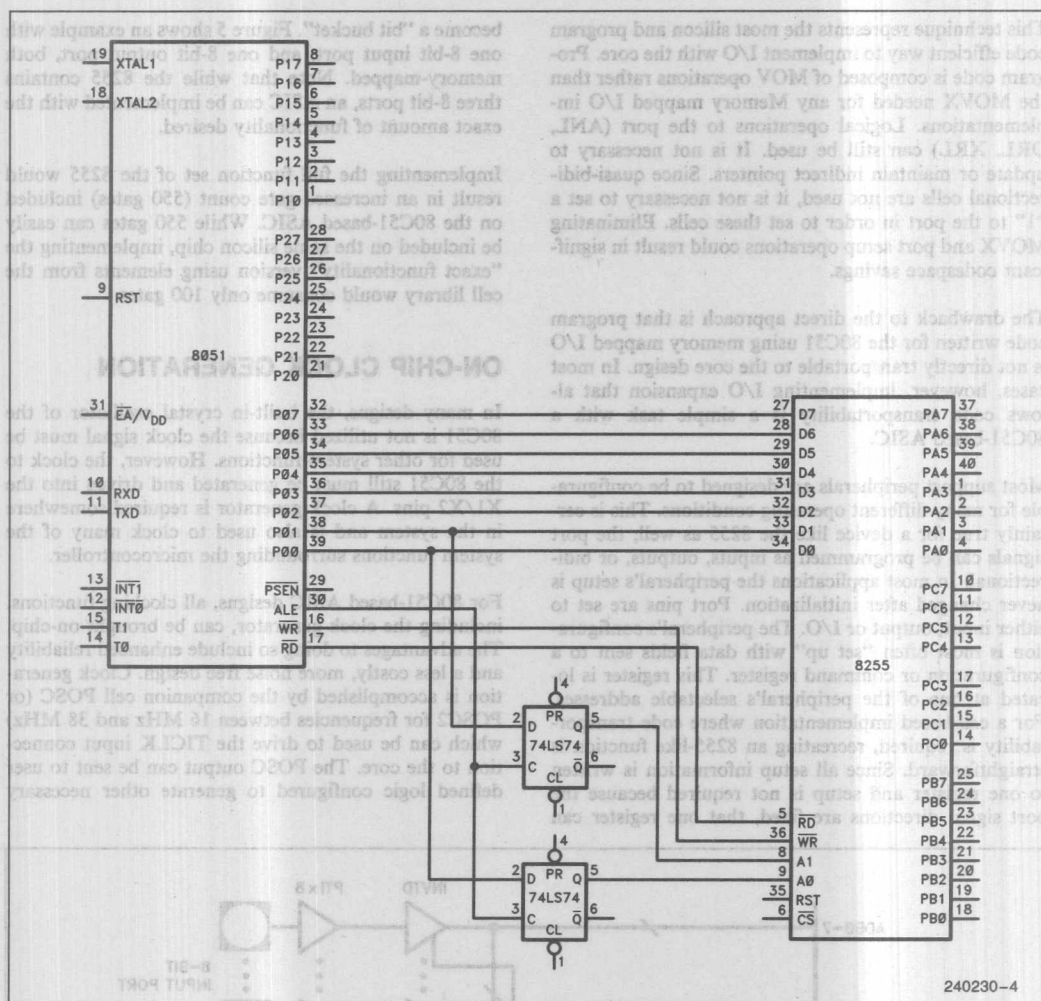


Figure 3. Simple MCS[®]-51 I/O Expansion with 8255

Figure 3 depicts one case where the 80C51 can gain an expanded set of I/O ports. In addition to requiring additional package pins, this implementation would require more power supply capacity and passive components (bypass capacitors) than would be necessary if the I/O expansion were to be included on-chip with the microcontroller. Not only is PC board size decreased, but the overall system reliability increases with the ASIC solution. In addition, the 8255 port expander, being a highly flexible device, requires software to configure the device to the application.

The simplest way to add I/O ports with the core is by way of a direct connection from the core's IP or OP signals through I/O functions selected from the cell library and connected to package pins. See Figure 4. In

this example, decoding is very simple and the component count is minimal.

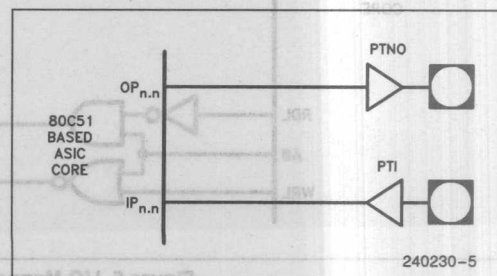


Figure 4. Direct Port Connections to ASIC Package Pins

code efficient way to implement I/O with the core. Program code is composed of MOV operations rather than the MOVX needed for any Memory mapped I/O implementations. Logical operations to the port (ANL, ORL, XRL) can still be used. It is not necessary to update or maintain indirect pointers. Since quasi-bidirectional cells are not used, it is not necessary to set a "1" to the port in order to set these cells. Eliminating MOVX and port setup operations could result in significant codespace savings.

The drawback to the direct approach is that program code written for the 80C51 using memory mapped I/O is not directly transportable to the core design. In most cases, however, implementing I/O expansion that allows code transportability is a simple task with a 80C51-based ASIC.

Most support peripherals are designed to be configurable for many different operating conditions. This is certainly true for a device like the 8255 as well; the port signals can be programmed as inputs, outputs, or bidirectional. In most applications the peripheral's setup is never changed after initialization. Port pins are set to either input, output or I/O. The peripheral's configuration is most often "set up" with data fields sent to a configuration or command register. This register is located at one of the peripheral's selectable addresses. For a cell-based implementation where code transportability is required, recreating an 8255-like function is straightforward. Since all setup information is written to one register and setup is not required because the port signal directions are fixed, that one register can

be brought on-chip. Figure 5 shows an example with one 8-bit input port, and one 8-bit output port, both memory-mapped. Note that while the 8255 contains three 8-bit ports, an ASIC can be implemented with the exact amount of functionality desired.

Implementing the full function set of the 8255 would result in an increased gate count (550 gates) included on the 80C51-based ASIC. While 550 gates can easily be included on the same silicon chip, implementing the "exact functionality" version using elements from the cell library would consume only 100 gates.

ON-CHIP CLOCK GENERATION

In many designs, the built-in crystal oscillator of the 80C51 is not utilized because the clock signal must be used for other system functions. However, the clock to the 80C51 still must be generated and driven into the X1/X2 pins. A clock generator is required somewhere in the system and is also used to clock many of the system functions surrounding the microcontroller.

For 80C51-based ASIC designs, all clocking functions, including the clock generator, can be brought on-chip. The advantages to doing so include enhanced reliability and a less costly, more noise free design. Clock generation is accomplished by the companion cell POSC (or POSC2 for frequencies between 16 MHz and 38 MHz) which can be used to drive the TCLK input connection to the core. The POSC output can be sent to user defined logic configured to generate other necessary

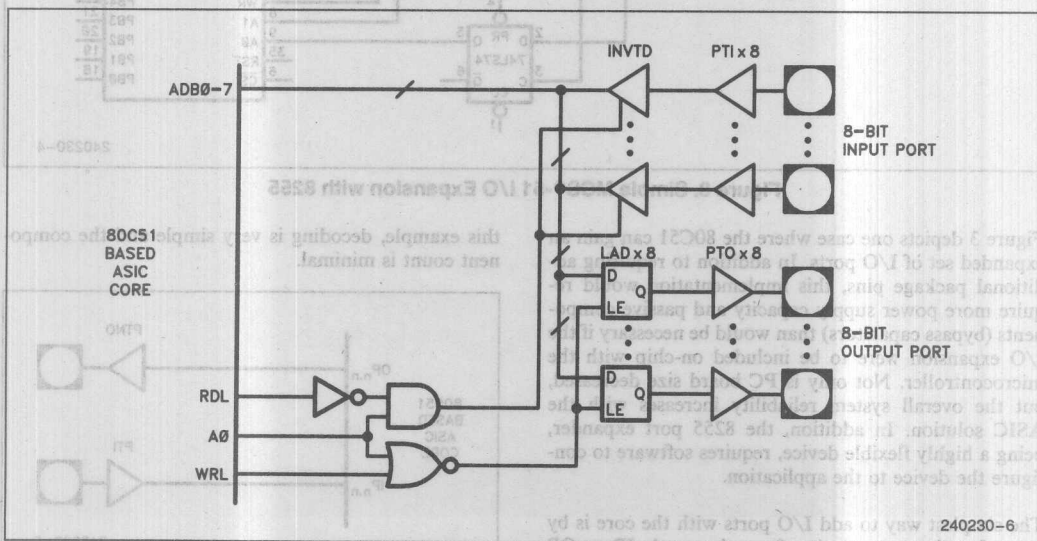


Figure 5. I/O Mapped Like Figure 3's 8255

clocks in a system. Where the POSC cell fan out is high and might cause concerns about clock edge skew, a cell like BUF2 can be used. For systems where it is required that the signal be brought off-chip (formerly off-board), the on-chip generated clocks can be sent to output cells and on to package pins. Figure 6 depicts generation flexibility and clock source for a 80C51-based ASIC design.

Figure 6 illustrates a design which requires a high frequency clock to operate on a section of user-defined logic. For this, cell POSC2 is selected for the ASIC design and is set to 24 MHz. In order to meet clock specifications for the core, this 24 MHz master clock is divided down in order to provide the required 12 MHz. As discussed in the 80C51-based ASIC data sheet, the ATE must be able to drive the core's clock directly. For test modes, the ATE-generated clock is driven to the core's TCLK connection.

Note that signal P2 is shown being used for the application's clocks. It is sent to other logic in the ASIC and to a package pin as well.

SHARING THE TEST BUS

In order for Automatic Test Equipment (ATE) to exercise the 80C51-based ASIC, the bus EADB must be brought directly to package pins. A specially designed I/O cell, PADB, must be directly connected to the EADB bus to ensure testability of the core as well as the user's logic.

Requiring the EADB bus to appear as package pins does not impose any design restrictions on 80C51-based ASICs designed to access external memory or peripherals. If your design does not call for the EADB to access external memory peripherals, the EADB may be multiplexed with user I/O. Contact your Intel Technology Center for the best implementation for your application.

Intel supplies all test programs required to completely test the ASIC core cell. Designers are required to supply test vectors that exercise their unique logic only.

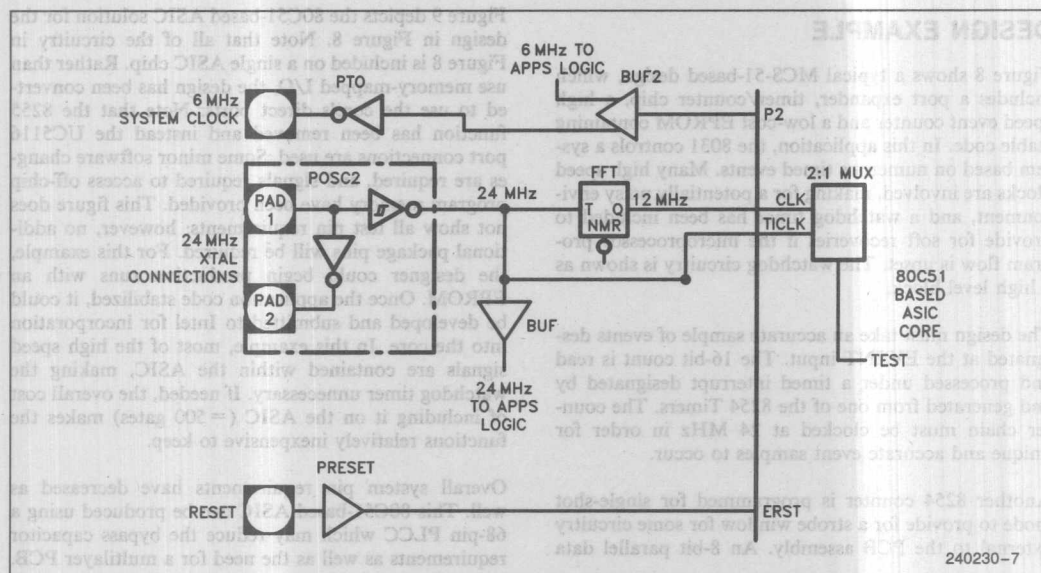


Figure 6. Integrating System Timing onto 80C51-Based ASIC

OBSERVING THE CONTENTS OF THE PROGRAM COUNTER

The 80C51-based ASIC core connections A0-A15 always display the contents of the program counter (except in the case of MOVX instructions.) This feature allows another level of real time control by monitoring instruction events within the core. By attaching comparator circuitry to the program counter contents, signals can be generated to depict events within the program. Figure 7 shows such a circuit.

The discussions in this note are not intended to be an exhaustive summary of the range of design possibilities available to the ASIC designer. Rather, it is hoped that it encourages the thought process toward even more innovative uses.

The following is an example of an actual system problem and how it was resolved using a 80C51-based ASIC. The example utilizes many of the techniques discussed above.

DESIGN EXAMPLE

Figure 8 shows a typical MCS-51-based design, which includes a port expander, timer/counter chip, a high speed event counter and a low-cost EPROM containing stable code. In this application, the 8031 controls a system based on numerous timed events. Many high speed clocks are involved, making for a potentially noisy environment, and a watchdog timer has been included to provide for soft recoveries if the microprocessor program flow is upset. The watchdog circuitry is shown as a high level block.

The design must take an accurate sample of events designated at the EVENT input. The 16-bit count is read and processed under a timed interrupt designated by and generated from one of the 8254 Timers. The counter chain must be clocked at 24 MHz in order for unique and accurate event samples to occur.

Another 8254 counter is programmed for single-shot mode to provide for a strobe window for some circuitry external to the PCB assembly. An 8-bit parallel data

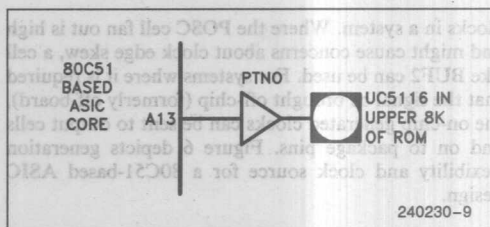


Figure 7. Signal which Designates when Program Execution is in Upper 8K ROM

input is required. The system must control peripherals external to this main assembly, resulting in a requirement for address decoder selection.

Note that adequate bypass capacitors are required due to the clock speeds and the high number of pin connections. (There are 272 pins, not including the WDT and Oscillator Blocks.) A multilayer PCB is also required to compensate for the amount of wires needed to connect all the components.

Figure 9 depicts the 80C51-based ASIC solution for the design in Figure 8. Note that all of the circuitry in Figure 8 is included on a single ASIC chip. Rather than use memory-mapped I/O, the design has been converted to use the core's direct ports. Note that the 8255 function has been removed and instead the UC5116 port connections are used. Some minor software changes are required, and signals required to access off-chip program memory have been provided. This figure does not show all test pin requirements; however, no additional package pins will be required. For this example, the designer could begin production runs with an EPROM. Once the application code stabilized, it could be developed and submitted to Intel for incorporation into the core. In this example, most of the high speed signals are contained within the ASIC, making the watchdog timer unnecessary. If needed, the overall cost of including it on the ASIC (= 500 gates) makes the functions relatively inexpensive to keep.

Overall system pin requirements have decreased as well. This 80C51-based ASIC can be produced using a 68-pin PLCC which may reduce the bypass capacitor requirements as well as the need for a multilayer PCB.

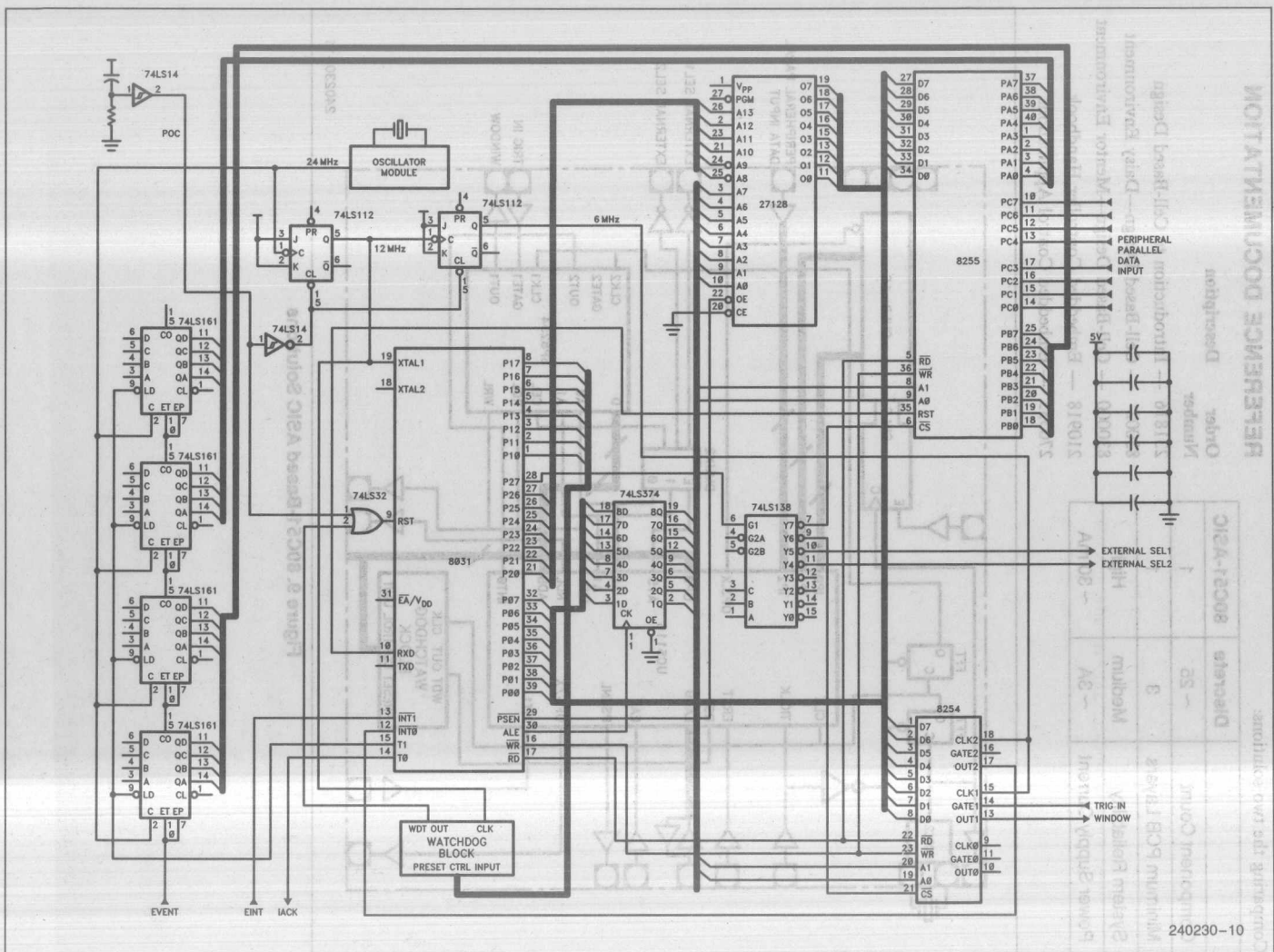


Figure 8. Typical MCS-51 Design, which Includes a Port Expander, Timer/Counter Chip, a High-Speed Event Counter, and a Low-Cost EPROM

Order Number	Description
231816	— Introduction to Cell-Based Design
83002	— Cell-Based Design—Daisy Environment
830000	— Cell-Based Design—Mentor Environment
210918	— Embedded Controller Handbook
270535	— Embedded Control Applications



A Fast-Turnaround, Easily Testable ASIC Chip for Serial Bus Control

Don Ellis and Shailesh Trivedi

Intel Corp.
Chandler, AZ

ABSTRACT

This paper describes the standard cell ASIC design methodology for a serial bus controller chip. This is a prototype CMOS chip which was designed in 19 weeks for an automotive application. The chip includes testability circuits which help attain 98% fault coverage.

INTRODUCTION

Fast-turnaround chip design has become important in the application-specific integrated circuit (ASIC) marketplace, where low production volumes preclude long design cycles. To address this market, the ASIC design methodology relies upon automatic layout software to generate fast chip layouts, at the expense of larger die sizes and somewhat lower performance. Pre-designed standard circuit cells eliminate the need for extensive circuit simulation, further shortening the design cycle. These design techniques can produce fast prototype chips for system demonstration and debug, or production parts for low-volume applications.

Intel's Automotive Operation in Chandler, Arizona recently employed standard cell ASIC technology to produce a prototype serial bus controller chip for an automotive customer. In this paper we will describe the design methodology used to meet the 19-week design schedule for this chip, along with the testability strategy which was implemented in order to achieve a 98% fault grade.

CHIP OVERVIEW AND CONSTRUCTION

The serial bus controller is a standard cell CMOS chip that interfaces a microprocessor to a serial communication bus in an automobile. The chip performs both transmit and receive functions. The transmit function consists of a first-in, first-out (FIFO) data buffer feeding a parallel-in, serial-out (PISO) shift register, and

the receive function consists of a serial-in, parallel-out (SIPO) shift register driving one port of a dual-port random access memory (DPRAM). The block diagram is shown in Figure 1.

The transmit function requires a decidedly non-standard 64 x 18 bit FIFO buffer. This is constructed with a 64 x 18 bit RAM and two address counters, as shown in Figure 2. The standard cell library did not contain a 64 x 18 bit RAM cell, so we had to construct it using an existing 64 x 8 RAM cell. We modified this cell, adding two more bits to create a 64 x 10 RAM cell, then connected it in parallel with the original 64 x 8 RAM, thus extending the word length to 18 bits. Before the 64 x 10 RAM cell could be added to the standard cell library, we had to fully characterize it using circuit simulation, like every other cell in the library. Two additional RC delay cells were also created to generate RAM read and write timings in the absence of microprocessor control signals.

The receive function requires a 1K x 8 bit dual-port RAM, but the standard cell library contained only single-port RAM cells. Fortunately, no cell modifications were necessary in this case. We used the existing 1K x 8 RAM cell, multiplexing its data and address buses to simulate dual-port operation, as shown in Figure 3. RAM read and write timings are once again generated using the RC delay cells mentioned above.

The final chip was manufactured in both single-layer metal (SLM) and double-layer metal (DLM) versions on a 1.5 micron CMOS process, resulting in a 355 x 294 mil chip with 68 I/O pins. It consists of 3 RAM arrays (9.3K bits total) and about 3,000 logic gates of control logic, for a total of 76,735 transistors. Of the 8,715 transistors contributed by the control logic, 11% belong to testability circuits which were added to increase the testability of the chip (i.e., shorten test program development time and tester run time). The testability strategy will be discussed later.

A Fast-Turnaround, Easily Testable ASIC

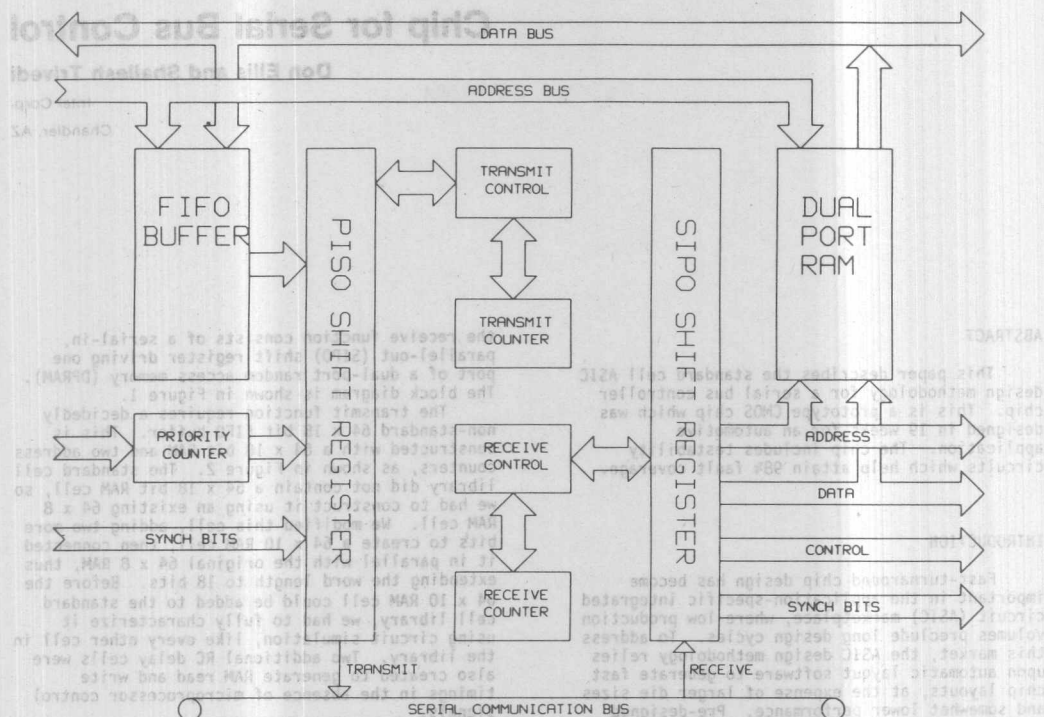


FIGURE 1. SERIAL BUS CONTROLLER BLOCK DIAGRAM

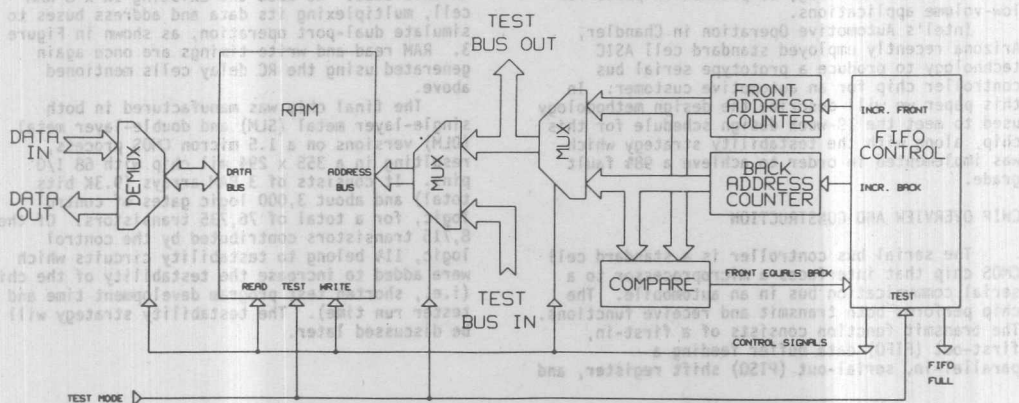


FIGURE 2. FIFO (FIRST-IN, FIRST-OUT) BUFFER

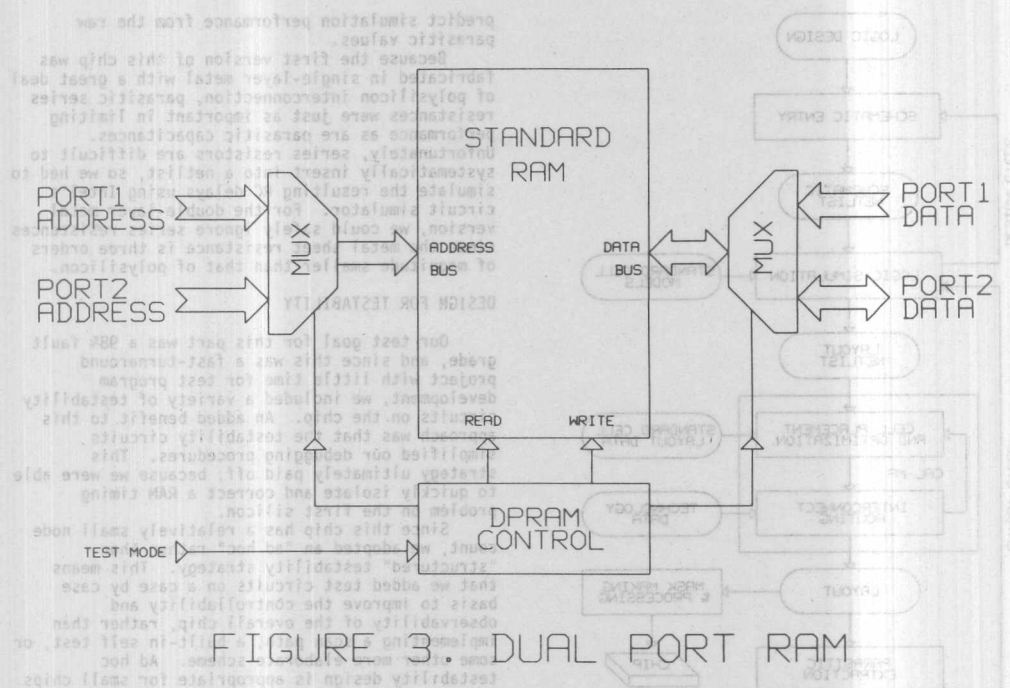


FIGURE 3. DUAL PORT RAM

STANDARD CELL DESIGN METHODOLOGY

The 19 week design schedule for this chip dictated the use of design automation tools. Since the chip included 3 large RAM arrays, gate arrays were impractical, so standard cells were used with automatic placement and routing software. The automatically generated layout was transferred into Intel's full custom design system for some final edits, and the usual design rule checking and verification procedures were followed prior to mask making and processing.

A standard cell design usually proceeds through the following steps:

1. Translation of the logic into standard cells.
2. Schematic capture into a computer database.
3. Extraction of a cell interconnection "netlist".
4. Logic and timing simulation.
5. Automatic layout generation.
6. Parasitic extraction and re-simulation.

The entire design procedure is outlined in Figure 4, and each step is described briefly below.

TRANSLATION INTO STANDARD CELLS

Our first task was to translate our customer's board-level schematics into a logic design consisting of subcircuits from the standard cell library. Since the customer's schematics referenced IC packages only, this involved the detailed design of the FIFO and DPRAM blocks (described above). A major part of the task was

the design of the extra standard cells mentioned above, with their characterization and inclusion in the cell library.

SCHEMATIC CAPTURE, NETLIST EXTRACTION, SIMULATION

We performed schematic capture on a Daisy Personal Logician (PC-AT based) workstation, where each of the standard cells was available as a basic circuit element. We "compiled" each schematic separately to verify its integrity, then linked them together into a complete design database. Finally, we generated a "netlist", or device interconnection list, from this database. This netlist served as input to Intel's logic simulator on our VAX, which we used to verify design correctness. The logic simulator flagged several timing and glitch problems which were corrected before proceeding to layout.

AUTOMATIC LAYOUT GENERATION

We performed layout generation using the CAL-MP program from Silver-Lisco. Working from the netlist, the program placed the three RAM arrays according to our instructions, then arranged the remaining standard cells in rows according to its own optimization algorithm. At this point prior to signal routing, we instructed the program to further iterate its optimization steps, as we manually modified several cell placements from the graphics terminal. Once all cell placements were determined, the program performed signal and power routing automatically.

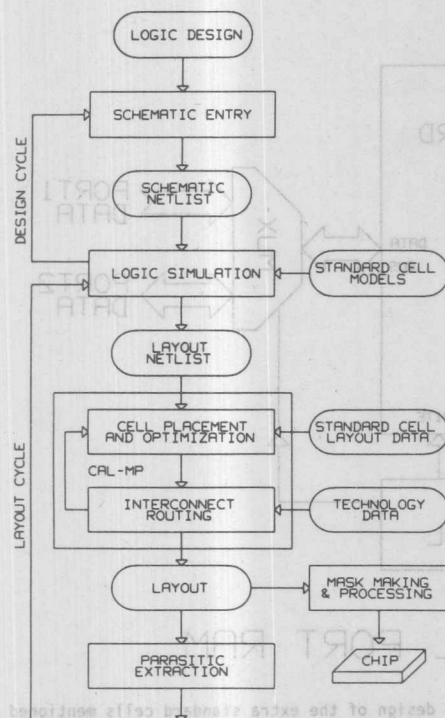


FIGURE 4. STANDARD CELL DESIGN FLOW

The CAL-MP program accepted layout constraints in a variety of forms. In addition to the netlist information, we defined pad placements and the number of standard cell rows, and constrained a few critical signals (such as clocks) to two vertical metal buses traversing the right and left sides of the chip. Furthermore, several unconstrained signals were assigned numerical "strengths" greater than the default of 1.0, which weighted their consideration in the optimization algorithm, tending to shorten them. We ultimately generated more than 20 layouts with widely varying signal strengths, until we were satisfied that very little further improvement was possible.

PARASTIC EXTRACTION

After a layout is generated, it must be proven to work in the presence of parasitic resistances and capacitances contributed by the signal interconnects. These parasitics are extracted from the layout and added to the netlist for a post-layout simulation cycle. In principle, each iterated layout should be re-simulated, but after about 10 layout generations we could easily

predict simulation performance from the raw parasitic values.

Because the first version of this chip was fabricated in single-layer metal with a great deal of polysilicon interconnection, parasitic series resistances were just as important in limiting performance as are parasitic capacitances. Unfortunately, series resistors are difficult to systematically insert into a netlist, so we had to simulate the resulting RC delays using Intel's circuit simulator. For the double-layer metal version, we could safely ignore series resistances since the metal sheet resistance is three orders of magnitude smaller than that of polysilicon.

DESIGN FOR TESTABILITY

Our test goal for this part was a 98% fault grade, and since this was a fast-turnaround project with little time for test program development, we included a variety of testability circuits on the chip. An added benefit to this approach was that the testability circuits simplified our debugging procedures. This strategy ultimately paid off, because we were able to quickly isolate and correct a RAM timing problem on the first silicon.

Since this chip has a relatively small node count, we adopted an "ad hoc" rather than "structured" testability strategy. This means that we added test circuits on a case by case basis to improve the controllability and observability of the overall chip, rather than implementing a scan path, a built-in self test, or some other more elaborate scheme. Ad hoc testability design is appropriate for small chips having relatively low transistor/pin ratios. This chip has 8,715 transistors (excluding those in the RAMs) versus 68 pins, for a transistor/pin ratio of 128. In contrast, Intel's 80386 microprocessor has 275,000 transistors versus 132 pins for a ratio of 2,083, clearly requiring structured testability techniques.

Two pins are allocated for test purposes, which are used to select among four modes: a normal operating mode, and three test modes. This test mode strategy is shown in Figure 5. The test modes are used to partition the chip into three isolated subcircuits to be tested independently. In each mode, signals with poor visibility internal to the active subcircuit are brought out to the pads, and the non-active subcircuits are turned off by disabling their clock inputs. The test program can then exercise the active circuit, with the goal of toggling each internal node for maximum fault coverage.

Eleven of the 28 chip inputs provide test inputs in the three test modes, and 16 of the 23 chip outputs serve double duty as test outputs. Although input pins can be connected to several internal test points in parallel (usually multiplexer inputs), only one signal at a time can drive an output pin. These outputs are multiplexed using three stages of 2:1 multiplexers (one for each mode), and the outputs are collected into a 16 bit "test bus" which circumnavigates the chip.

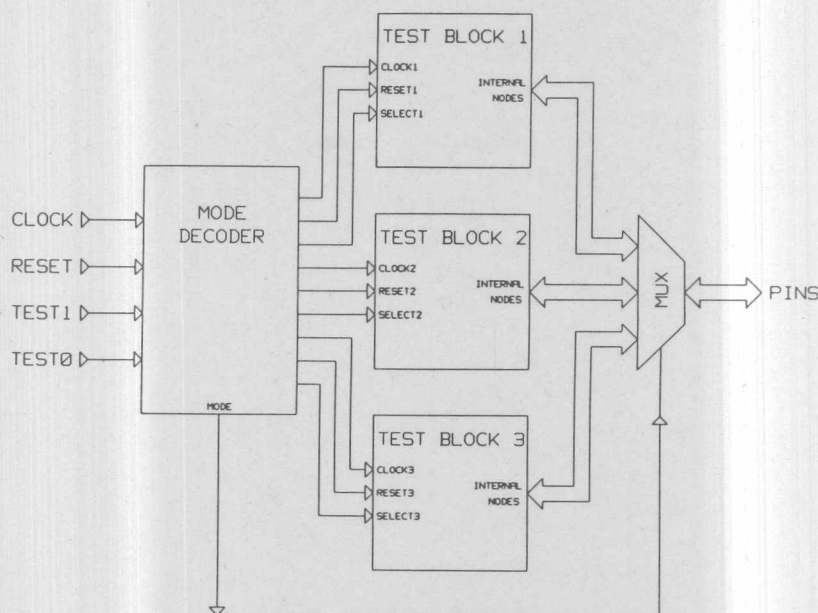


FIGURE 5. TEST MODE STRATEGY

RAM TESTABILITY

RAM testability is a special case, because a RAM is inherently fully testable provided its data and address buses are accessible, along with the necessary control signals. The difficulty here is that the RAMs are embedded in functional blocks which, especially in the FIFO, tends to disguise the inherent RAM accessibility.

Inside the DPRAM, the 1K x 8 RAM is read directly by the microprocessor using the external data and address buses, so observability is no problem. Writes, however, occur from the SIPO during serial reception. It would be particularly painful to test a 1K RAM using serial writes, so a modification was necessary to improve RAM controllability. In test mode, the address multiplexer is held to the address bus by overriding the select line, and a set of eight extra multiplexers were added to the data demultiplexer to allow bidirectional data flow into and out of the RAM. Thus, the SIPO circuit is completely bypassed in test mode.

The FIFO RAM is addressed by one of two counters in the operating mode, which presents a problem unless we are willing to accept sequential test addressing, or at least a very complex address setup procedure. The solution was to override the address bus with a multiplexer fed by input pin signals. The data bus presented the same problem as the DPRAM, but in reverse: data

is written by the microprocessor using the external buses, but reads are serialized in the PISO. We decided that serial output was acceptable in FIFO test mode, however, because the FIFO has only 64 locations to test (versus 1024 in the DPRAM), and the words are 18 bits long, which would require 18 extra multiplexers. Thus, for the FIFO data, we left well enough alone.

CONCLUSION

This serial bus controller chip was designed using ASIC techniques in a very short time, resulting in a quick prototype chip which our automotive customer could use to evaluate his system design in a timely manner. The inclusion of testability circuits further shortened the engineering debug time as well as the manufacturing test time. This project demonstrates that standard cell design is an attractive fast-turnaround methodology, and that a good testability strategy provides additional benefits which outweigh the extra design effort.

ACKNOWLEDGEMENT

The authors would like to thank Graham Tubbs, for guiding us through the maze of ASIC design tools, Dinesh Maheshwari and Keith Steele, who helped prepare our final layout for processing, and Mukund Patel and Magdiel Galan who helped us test and debug the final chip.

UPI-452 Accelerates iAPX 286 Bus Performance

CHRISTOPHER SCOTT
TECHNICAL MARKETING ENGINEER
INTEL CORPORATION

INTRODUCTION

The UPI-452 targets the leading problem in peripheral to host interfacing, the interface of a slow peripheral with a fast Host or "bus utilization". The solution is data buffering to reduce the delay and overhead of transferring data between the Host microprocessor and I/O subsystem. The Intel CMOS UPI-452 solves this problem by combining a sophisticated programmable FIFO buffer and a slave interface with an MSC-51 based microcontroller.

The UPI-452 is Intel's newest Universal Peripheral Interface family member. The UPI-452 FIFO buffer enables Host-peripheral communications to be through streams or bursts of data rather than by individual bytes. In addition the FIFO provides a means of embedding commands within a stream or block of data. This enables the system designer to manage data and commands to further off-load the Host.

The UPI-452 interfaces to the iAPX 286 microprocessor as a standard Intel slave peripheral device. READ, WRITE, CS and address lines from the Host are used to access all of the Host addressable UPI-452 Special Function Registers (SFR).

The UPI-452 combines an MSC-51 microcontroller, with 256 bytes of on-chip RAM and 8K bytes of EPROM/ROM, twice that of the 80C51, a two channel DMA controller and a sophisticated 128 byte, two channel, bidirectional FIFO in a single device. The UPI-452 retains all of the 80C51 architecture, and is fully compatible with the MSC-51 instruction set.

This application note is a description of an iAPX 286 to UPI-452 slave interface. Included is a discussion of the respective timings and design considerations. This application note is meant as a supplement to the UPI-452 Advance Data Sheet. The user should consult the data sheet for additional details on the various UPI-452 functions and features.

UPI-452 iAPX 286 SYSTEM CONFIGURATION

The interface described in this application note is shown in Figure 1, iAPX 286 UPI-452 System Block Diagram. The iAPX 286 system is configured in a local bus architecture design. DMA between the Host and the UPI-452 is supported by the 82258 Advanced DMA Controller. The Host microprocessor accesses all UPI-452 externally addressable registers through address decoding (see Table 3, UPI-452 External Address Decoding). The timings and interface descriptions below are given in equation form with examples of specific calculations. The goal of this application note is a set of interface analysis equations. These equations are the tools a system designer can use to fully utilize the features of the UPI-452 to achieve maximum system performance.

HOST-UPI-452 FIFO SLAVE INTERFACE

The UPI-452 FIFO acts as a buffer between the external Host 80286 and the internal CPU. The FIFO allows the Host - peripheral interface to achieve maximum decoupling of the interface. Each of the two FIFO channels is fully user programmable. The FIFO buffer ensures that the respective CPU, Host or internal CPU, receives data in the same order as transmitted. Three slave bus interface handshake methods are supported by the UPI-452; DMA, Interrupt and Polled.

The interface between the Host 80286 and the UPI-452 is accomplished with a minimum of signals. The 8 bit data bus plus READ, WRITE, CS, and A0-2 provide access to all of the externally addressable UPI-452 registers including the two FIFO channels. Interrupt and DMA handshaking pins are tied directly to the interrupt controller and DMA controller respectively.

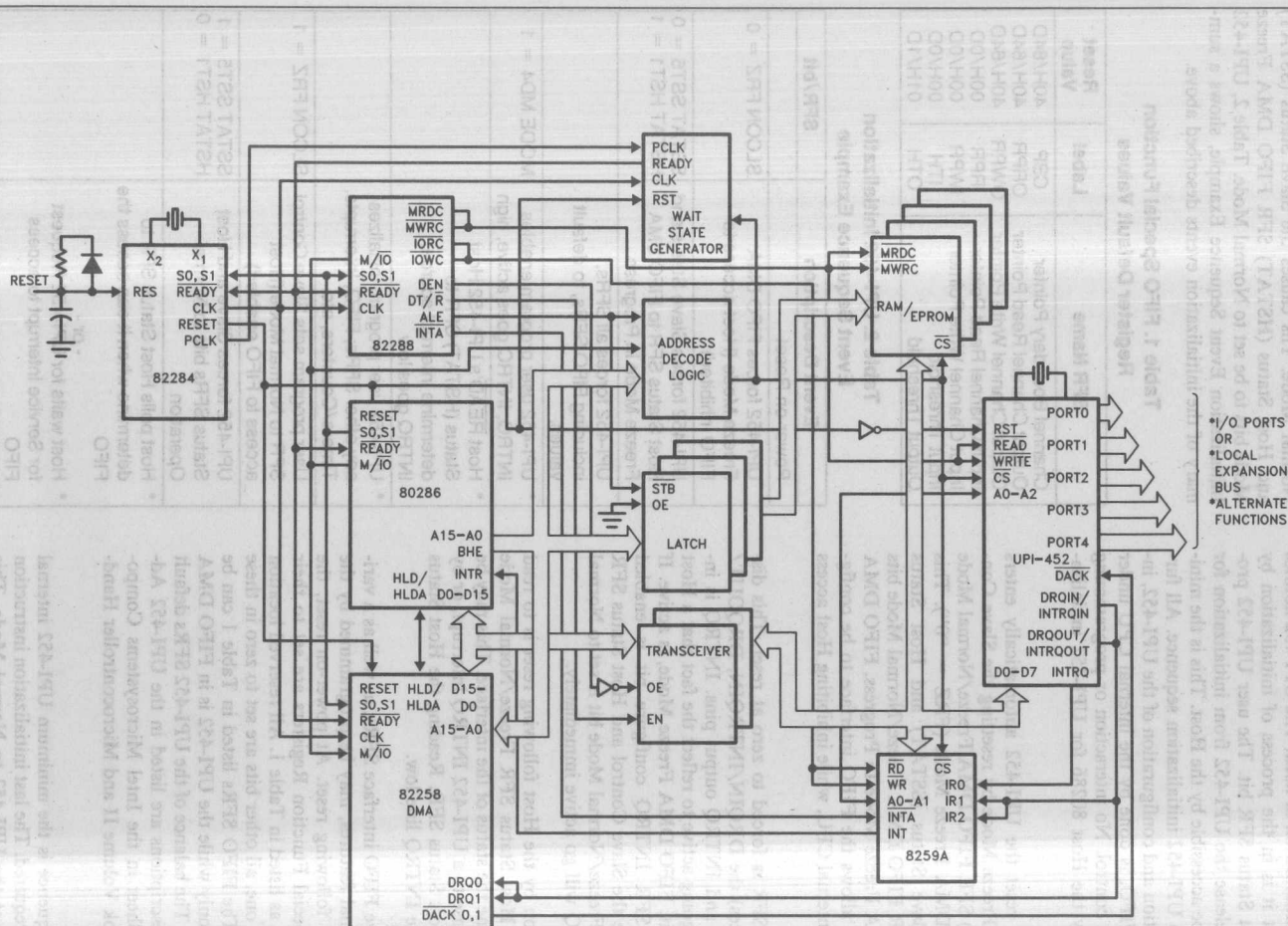
DMA transfers between the Host and UPI-452 are controlled by the Host processors DMA controller. In the example shown in Figure 1, the Host DMA controller is the 82258 Advanced DMA Controller. An internal DMA transfer to or from the FIFO, as well as between other internal elements, is controlled by the UPI-452 internal DMA processor. The internal DMA processor can also transfer data between Input and Output FIFO channels directly. The description that follows details the UPI-452 interface from both the Host processor's and the UPI-452's internal CPU perspective.

One of the unique features of the UPI-452 FIFO is its ability to distinguish between commands and data embedded in the same data block. Both interrupts and status flags are provided to support this operation in either direction of data transfer. These flags and interrupts are triggered by the FIFO logic independent of, and transparent to either the Host or internal CPUs. Commands embedded in the data block, or stream, are called Data Stream Commands.

Programmable FIFO channel Thresholds are another unique feature of the UPI-452. The Thresholds provide for interrupting the Host only when the Threshold number of bytes can be read or written to the FIFO buffer. This further decouples the Host UPI-452 interface by relieving the Host of polling the buffer to determine the number of bytes that can be read or written. It also reduces the chances of overrun and underrun errors which must be processed.

The UPI-452 also provides a means of bypassing the FIFO, in both directions, for an immediate interrupt of either the Host or internal CPU. These commands are called Immediate Commands. A complete description of the internal FIFO logic operation is given in the FIFO Data Structure section.

Figure 1. APX 286 UPI-452 System Block Diagram



292018-1

UPI-452 INITIALIZATION

The UPI-452 at power-on reset automatically performs a minimum initialization of itself. The UPI-452 notifies the Host that it is in the process of initialization by setting a Host Status SFR bit. The user UPI-452 program must release the UPI-452 from initialization for the FIFO to be accessible by the Host. This is the minimum Host to UPI-452 initialization sequence. All further initialization and configuration of the UPI-452, including the FIFO, is done by the internal CPU under user program control. No interaction or programming is required by the Host 80286 for UPI-452 initialization.

At power-on reset the UPI-452 automatically enters FIFO DMA Freeze Mode by resetting the Slave Control (SLCON) SFR FIFO DMA Freeze/Normal Mode bit to FIFO DMA Freeze Mode (FRZ = "0"). This forces the Slave Status (SSTAT) and Host Status (HSTAT) SFR FIFO DMA Freeze/Normal Mode bits to FIFO DMA Freeze Mode In Progress. FIFO DMA Freeze Mode allows the FIFO interface to be configured, by the internal CPU, while inhibiting Host access to the FIFO.

The MODE SFR is forced to zero at reset. This disables, (tri-states) the DRQIN/INTRQIN, DRQOUT/INTRQOUT and INTRQ output pins. INTRQ is inhibited from going active to reflect the fact that a Host Status SFR bit, FIFO DMA Freeze Mode, is active. If the MODE SFR INTRQ configure bit is enabled (= '1'), before the Slave Control and Host Status SFR FIFO DMA Freeze/Normal Mode bit is set to Normal Mode, INTRQ will go active immediately.

The first action by the Host following reset is to read the UPI-452 Host Status SFR Freeze/Normal Mode bit to determine the status of the interface. This may be done in response to a UPI-452 INTRQ interrupt, or by polling the Host Status SFR. Reading the Host Status SFR resets the INTRQ line low.

Any of the five FIFO interface SFRs, as well as a variety of additional features, may be programmed by the internal CPU following reset. At power-on reset, the five FIFO Special Function Registers are set to their default values as listed in Table 1. All reserved location bits are set to one, all other bits are set to zero in these three SFRs. The FIFO SFRs listed in Table 1 can be programmed only while the UPI-452 is in FIFO DMA Freeze Mode. The balance of the UPI-452 SFRs default values and descriptions are listed in the UPI-452 Advance Data Sheet in the Intel Microsystems Component Handbook Volume II and Microcontroller Handbook.

The above sequence is the minimum UPI-452 internal initialization required. The last initialization instruction must always set the UPI-452 to Normal Mode. This causes the UPI-452 to exit Freeze Mode and enables

Host read/write access of the FIFO. The internal CPU sets the Slave Control (SLCON) SFR FIFO DMA Freeze/Normal Mode (FRZ) bit high (= 1) to activate Normal Mode. This causes the Slave Status (SSTAT) and Host Status (HSTAT) SFR FIFO DMA Freeze Mode bits to be set to Normal Mode. Table 2, UPI-452 Initialization Event Sequence Example, shows a summary of the initialization events described above.

Table 1. FIFO Special Function Register Default Values

SFR Name	Label	Reset Value
Channel Boundary Pointer	CBP	40H/64D
Output Channel Read Pointer	ORPR	40H/64D
Output Channel Write Pointer	OWPR	40H/64D
Input Channel Read Pointer	IRPR	00H/0D
Input Channel Write Pointer	IWPR	00H/0D
Input Threshold	ITH	00H/0D
Output Threshold	OTH	01H/1D

Table 2. UPI-452 Initialization Event Sequence Example

Event Description	SFR/bit
Power-on Reset	
UPI-452 forces FIFO DMA Freeze Mode (Host access to FIFO inhibited)	SLCON FRZ = 0
UPI-452 forces Slave Status and Host Status SFR to FIFO DMA Freeze Mode In Progress	SSTAT SST5 = 0 HSTAT HST1 = 1
UPI-452 forces all SFRs, including FIFO SFRs, to default values.	
* UPI-452 user program enables INTRQ, INTRQ goes active, high	MODE MD4 = 1
* Host READ's UPI-452 Host Status (HSTAT) SFR to determine interrupt source, INTRQ goes low	
* UPI-452 user program initializes any other SFRs; FIFO, Interrupts, Timers/Counters, etc.	
User program sets Slave Control SFR to Normal Mode (Host access to FIFO enabled)	SLCON FRZ = 1
UPI-452 forces Slave and Host Status SFRs bits to Normal Operation	SSTAT SST5 = 1 HSTAT HST1 = 0
* Host polls Host Status SFR to determine when it can access the FIFO	
- or -	
* Host waits for UPI-452 Request for Service interrupt to access FIFO	

* user option

FIFO DATA STRUCTURES

Overview

The UPI-452 provides three means of communication between the Host microprocessor and the UPI-452 in either direction;

- Data
- Data Stream Commands
- Immediate Commands

Data and Data Stream Commands (DSC) are transferred between the Host and UPI-452 through the UPI-452 FIFO buffer. The third, Immediate Commands, provides a means of bypassing the FIFO entirely. These three data types are in addition to direct access by either Host or Internal CPU of dedicated Status and Control Special Function Registers (SFR).

The FIFO appears to both the Host 80286 and the internal CPU as 8 bits wide. Internally the FIFO is logically nine bits wide. The ninth bit indicates whether the byte is a data or a Data Stream Command (DSC) byte; 0 = data, 1 = DSC. The ninth bit is set by the FIFO logic in response to the address specified when writing to the FIFO by either Host or internal CPU. The FIFO uses the ninth bit to condition the UPI-452 interrupts and status flags as a byte is made available for a Host or internal CPU read from the FIFO. Figures 2 and 3 show the structure of each FIFO channel and the logical ninth bit.

It is important to note that both data and DSCs are actually entered into the FIFO buffer (see Figures 2 and 3). External addressing of the FIFO determines the state of the internal FIFO logic ninth bit. Table 3 shows the UPI-452 External Address Decoding used by the Host and the corresponding action. The internal CPU interface to the FIFO is essentially identical to the external Host interface. Dedicated internal Special Function Registers provide the interface between the FIFO, internal CPU and the internal two channel DMA processor. FIFO read and write operations by the Host and internal CPU are interleaved by the UPI-452 so they appear to be occurring simultaneously.

The ninth bit provides a means of supporting two data types within the FIFO buffer. This feature enables the Host and UPI-452 to transfer both commands and data while maintaining the decoupled interface a FIFO buffer provides. The logical ninth bit provides both a means of embedding commands within a block of data and a means for the internal CPU, or external Host, to discriminate between data and commands. Data or DSCs may be written in any order desired. Data Stream

Commands can be used to structure or dispatch the data by defining the start and end of data blocks or packets, or how the data following a DSC is to be processed.

A Data Stream Command (DSC) acts as an internal service routine vector. The DSC generates an interrupt to a service routine which reads the DSC. The DSC byte acts as an address vector to a user defined service routine. The address can be any program or data memory location with no restriction on the number of DSCs or address boundaries.

A Data Stream Command (DSC) can also be used to clear data from the FIFO or "FLUSH" the FIFO. This is done by appending a DSC to the end of a block of data entered in the FIFO which is less than the programmed threshold number of bytes. The DSC will cause an interrupt, if enabled, to the respective receiving CPU. This ensures that a less than Threshold number of bytes in the FIFO will be read. Two conditions force a Request for Service interrupt, if enabled, to the Host. The first is due to a Threshold number of bytes having been written to the FIFO Output channel; the second is if a DSC is written to the Output FIFO channel. If less than the Threshold number of bytes are written to the Output FIFO channel, the Host Status SFR flag will not be set, and a Request for Service interrupt will not be generated, if enabled. By appending a DSC to end of the data block, the FIFO Request for Service flag and/or interrupt will be generated.

An example of a FIFO Flush application is a mass storage subsystem. The UPI-452 provides the system interface to a subsystem which supports tape and disk storage. The FIFO size is dynamically changed to provide the maximum buffer size for the direction of transfer. Large data blocks are the norm in this application. The FIFO Flush provides a means of purging the FIFO of the last bytes of a transfer. This guarantees that the block, no matter what its size, will be transmitted out of the FIFO.

Immediate Commands allow more direct communication between the Host processor and the UPI-452 by bypassing the FIFO in either direction. The Immediate Command IN and OUT SFRs are two more unique address locations externally and internally addressable. Both DSCs and Immediate Commands have internal interrupts and interrupt priorities associated with their operation. The interrupts are enabled or disabled by setting corresponding bits in the Slave Control (SLCON), Interrupt Enable (IEC), Interrupt Priority (IPC) and Interrupt Enable and Priority (IEP) SFRs. A detailed description of each of these may be found in the UPI-452 Advance Information Data Sheet.

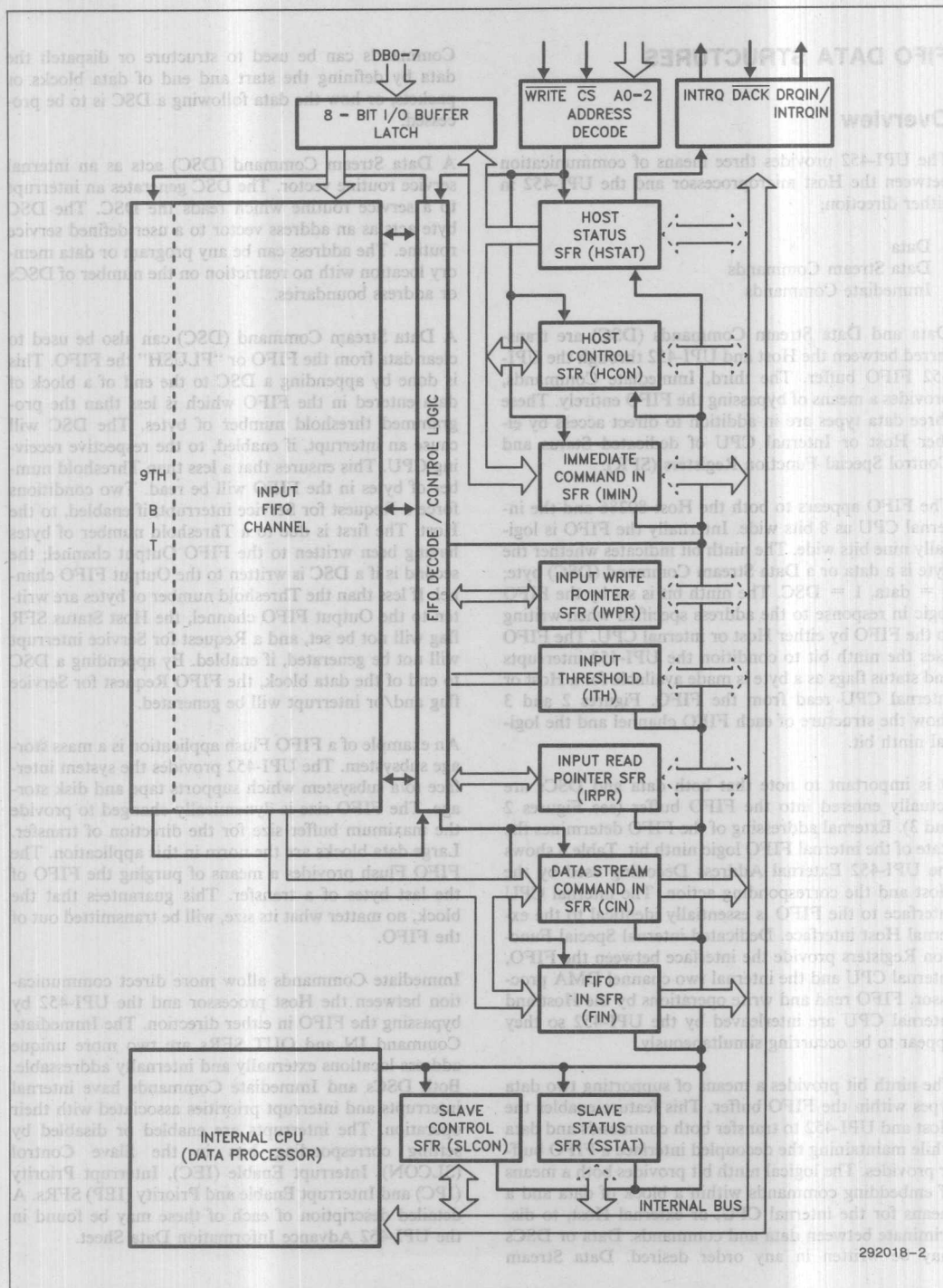
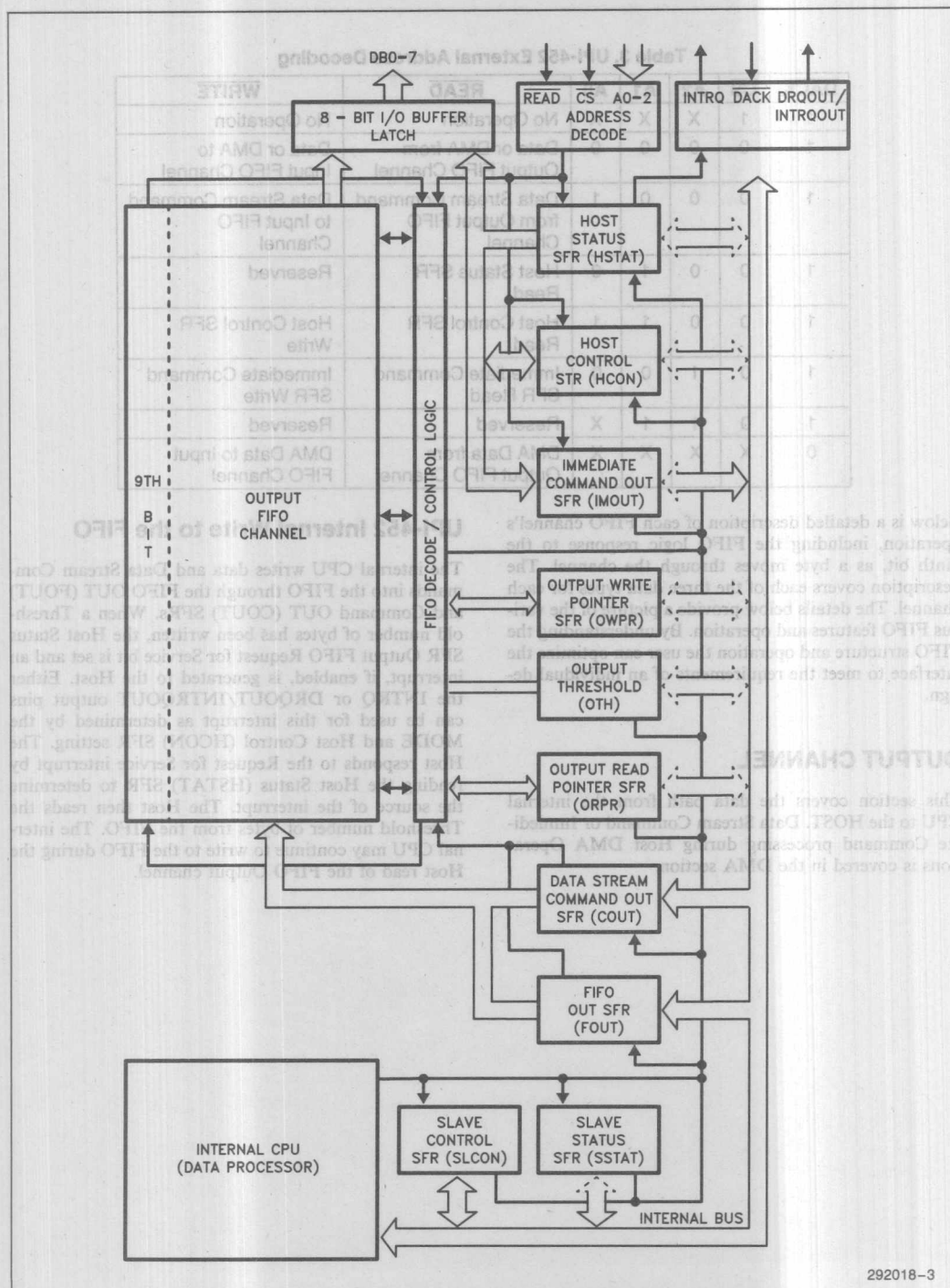


Figure 2. Input FIFO Channel Functional Diagram



292018-3

Table 3. UPI-452 External Address Decoding

DAK	CS	A2	A1	A0	READ	WRITE
1	1	X	X	X	No Operation	No Operation
1	0	0	0	0	Data or DMA from Output FIFO Channel	Data or DMA to Input FIFO Channel
1	0	0	0	1	Data Stream Command from Output FIFO Channel	Data Stream Command to Input FIFO Channel
1	0	0	1	0	Host Status SFR Read	Reserved
1	0	0	1	1	Host Control SFR Read	Host Control SFR Write
1	0	1	0	0	Immediate Command SFR Read	Immediate Command SFR Write
1	0	1	1	X	Reserved	Reserved
0	X	X	X	X	DMA Data from Output FIFO Channel	DMA Data to Input FIFO Channel

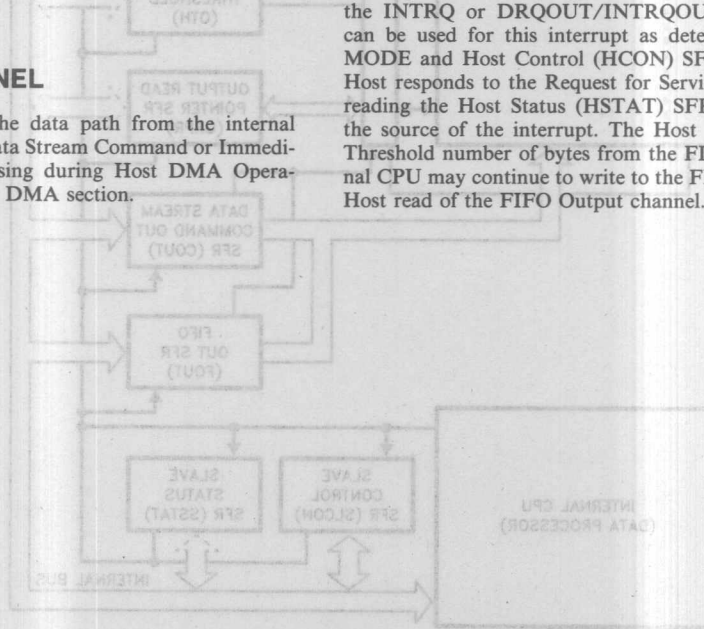
Below is a detailed description of each FIFO channel's operation, including the FIFO logic response to the ninth bit, as a byte moves through the channel. The description covers each of the three data types for each channel. The details below provide a picture of the various FIFO features and operation. By understanding the FIFO structure and operation the user can optimize the interface to meet the requirements of an individual design.

OUTPUT CHANNEL

This section covers the data path from the internal CPU to the HOST. Data Stream Command or Immediate Command processing during Host DMA Operations is covered in the DMA section.

UPI-452 Internal Write to the FIFO

The internal CPU writes data and Data Stream Commands into the FIFO through the FIFO OUT (FOUT) and Command OUT (COUT) SFRs. When a Threshold number of bytes has been written, the Host Status SFR Output FIFO Request for Service bit is set and an interrupt, if enabled, is generated to the Host. Either the INTRQ or DRQOUT/INTRQOUT output pins can be used for this interrupt as determined by the MODE and Host Control (HCON) SFR setting. The Host responds to the Request for Service interrupt by reading the Host Status (HSTAT) SFR to determine the source of the interrupt. The Host then reads the Threshold number of bytes from the FIFO. The internal CPU may continue to write to the FIFO during the Host read of the FIFO Output channel.



Data Stream Commands may be written to the Output FIFO channel at any time during a write of data bytes. The write instruction need only specify the Command Out (COUT) SFR in the direct register instruction used. Immediate Commands may also be written at any time to the Immediate Command OUT (IMOUT) SFR. The Host reads Immediate Commands from the Immediate Command OUT (IMOUT).

The internal CPU can determine the number of bytes to write to the FIFO Output channel in one of three ways. The first, and most efficient, is by utilizing the internal DMA processor which will automatically manage the writing of data to avoid Underrun or Overrun Errors. The second is for the internal CPU to read the Output FIFO channels Read and Write Pointers and compare their values to determine the available space. The third method for determining the available FIFO space is to always write the programmed channel size number of bytes to the Output FIFO. This method would use the Overrun Error flag and interrupt to halt FIFO writing whenever the available space was less than the channel size. The interrupt service routine could read the channel pointers to determine or monitor the available channel space. The time required for the internal CPU to write data to the Output FIFO channel is a function of the individual instruction cycle time and the number of bytes to be written.

Host Read from the FIFO

The Host reads data or Data Stream Commands (DSC) from the FIFO in response to the Host Status (HSTAT) SFR flags and interrupts, if enabled. All Host read operations access the same UPI-452 internal I/O Buffer Latch. At the end of the previous Host FIFO read cycle a byte is loaded from the FIFO into the I/O Buffer Latch and Host Status (HSTAT) SFR bit 5 is set or cleared (1 = DSC, 0 = data) to reflect the state of the byte's FIFO ninth bit. If the FIFO ninth bit is set (= 1) indicating a DSC, an interrupt is generated to the external Host via INTRQ pin or INTRQIN/INTRQOUT pins as determined by Host Control (HCON) SFR bit 1. The Host then reads the Host Status (HSTAT) SFR to determine the source of the interrupt.

The most efficient Host read operation of the FIFO Output channel is through the use of Host DMA. The UPI-452 fully supports external DMA handshaking. The MODE and Host Control SFRs control the configuration of UPI-452 Host DMA handshake outputs. If Host DMA is used the Threshold Request for Service interrupt asserts the UPI-452 DMA Request (DRQOUT) output. The Host DMA processor acknowledges with \overline{DACK} which acts as a chip select of the FIFO channels. The DMA transfer would stop when either the threshold byte count had been read, as programmed in the Host DMA processor, or when the DRQOUT output is brought inactive by the UPI-452.

INPUT CHANNEL

This section covers the data path from the HOST to the internal CPU or internal DMA processor. The details of Data Stream Command or Immediate Command processing during internal DMA operations are covered in the DMA section below.

Host Write to the FIFO

The Host writes data and Data Stream Commands into the FIFO through the FIFO IN (FIN) and Command IN (CIN) SFRs. When a Threshold number of bytes has been read out of the Input FIFO channel by the internal CPU, the Host Status SFR Input FIFO Request for Service bit is set and an interrupt, if enabled, is generated to the Host. The Input FIFO Threshold interrupt tells the Host that it may write the next block of data into the FIFO. Either the INTRQ or DRQIN/INTRQIN output pins can be used for this interrupt as determined by the MODE and Host Control (HCON) SFR settings. The Host may continue to write to the FIFO Input channel during the internal CPU read of the FIFO. Data Stream Commands may be written to the FIFO Input channel at any time during a write of data bytes. Immediate Commands may also be written at any time to the Immediate Command IN (IMIN) SFR.

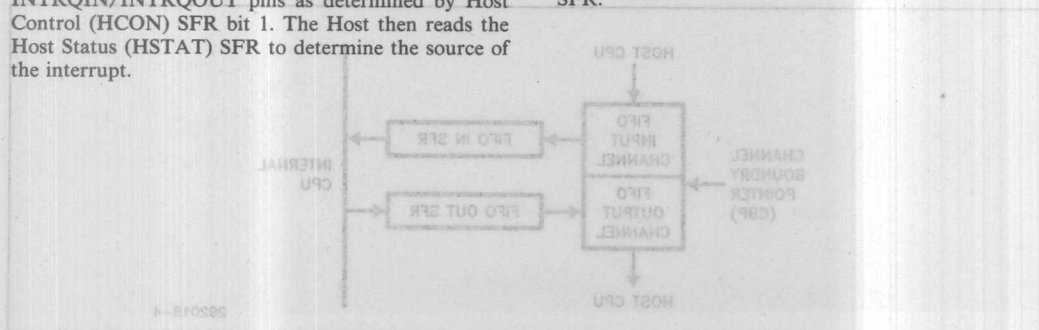


Figure 4: Full Duplex FIFO Operation

The Host also has three methods for determining the available FIFO space. Two are essentially identical to that of the internal CPU. They involve reading the FIFO Input channel pointers and using the Host Status SFR Underrun and Overrun Error flags and Request for Service interrupts these would generate, if enabled in combination. The third involves using the UPI-452 Host DMA controller handshake signals and the programmed Input FIFO Threshold. The Host would receive a Request for Service interrupt when an Input FIFO channel has a Threshold number of bytes able to be written by the Host. The Host service routine would then write the Threshold number of bytes to the FIFO.

If a Host DMA is used to write to the FIFO Input channel, the Threshold Request for Service interrupt could assert the UPI-452 DRQIN output. The Host DMA processor would assert DACK and the FIFO write would be completed by Host the DMA processor. The DMA transfer would stop when either the Threshold byte count had been written or the DRQIN output was removed by the UPI-452. Additional details on Host and internal DMA operation is given below.

Internal Read of the FIFO

At the end of an internal CPU read cycle a byte is loaded from the FIFO buffer into the FIFO IN/Command IN SFR and Slave Status (SSTAT) SFR bit 1 is set or cleared (1 = data, 0 = DSC) to reflect the state of the FIFO ninth bit. If the byte is a DSC, the FIFO ninth bit is set (= 1) and an interrupt is generated, if enabled, to the Internal CPU. The internal CPU then reads the Slave Status (SSTAT) SFR to determine the source of the interrupt.

Immediate Commands are written by the Host and read by the internal CPU through the Immediate Command IN (IMIN) SFR. Once written, an Immediate Command sets the Slave Status (SSTAT) SFR flag bit and generates an interrupt, if enabled, to the internal CPU. In response to the interrupt the internal CPU

reads the Slave Status (SSTAT) SFR to determine the source of the interrupt and service the Immediate Command.

FIFO INPUT/OUTPUT CHANNEL SIZE

Host

The Host does not have direct control of the FIFO Input or Output channel sizes or configuration. The Host can, however, issue Data Stream Commands or Immediate Commands to the UPI-452 instructing the UPI-452 to reconfigure the FIFO interface by invoking FIFO DMA Freeze Mode. The Data Stream Command or Immediate Command would be a vector to a service routine which performs the specific reconfiguration.

UPI-452 Internal

The default power-on reset FIFO channel sizes are listed in the "Initialization" section and can be set only by the internal CPU during FIFO DMA Freeze Mode. The FIFO channel size is selected to achieve the optimum application performance. The entire 128 byte FIFO can be allocated to either the Input or Output channel. In this case the other channel consists of a single SFR; FIFO IN/Command IN or FIFO OUT/Command OUT SFR. Figure 4 shows a FIFO division with a portion devoted to each channel. Figure 5 shows a FIFO configuration with all 128 bytes assigned to the Output channel.

The FIFO channel Threshold feature allows the user to match the FIFO channel size and the performance of the internal and Host data transfer rates. The programmed Threshold provides an elasticity to the data transfer operation. An example is if the Host FIFO

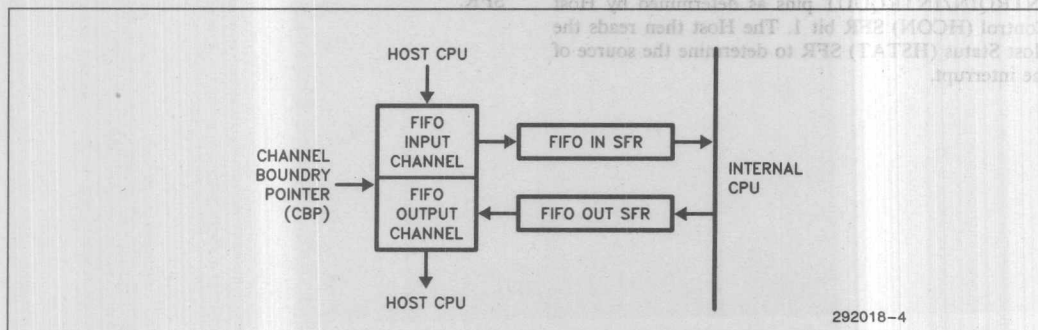


Figure 4. Full Duplex FIFO Operation

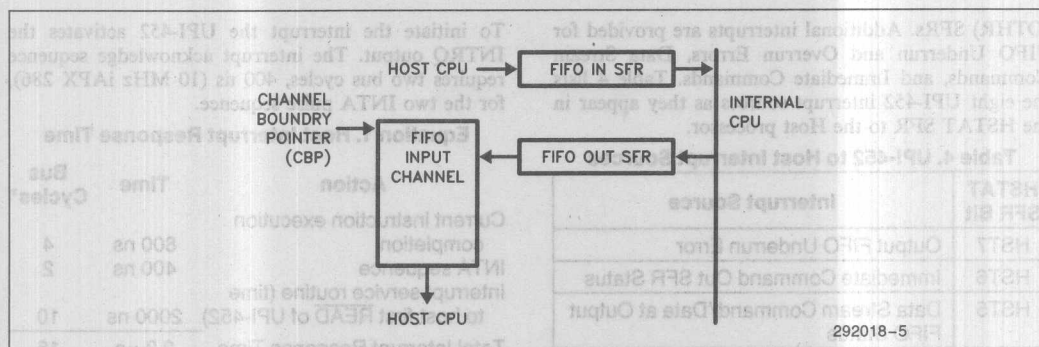


Figure 5. Entire FIFO Assigned to Output Channel

data transfer rate is twice as fast as the internal FIFO DMA data transfer rate. In this example the FIFO Input channel size is programmed to be 64 bytes and the Input channel Threshold is programmed to be 20 bytes. The Host writes the first 64 bytes to the Input FIFO. When the internal DMA processor has read 20 bytes the Threshold interrupt, or DMA request (DRQIN), is generated to signal the Host to begin writing more data to the Input FIFO channel. The internal DMA processor continues to read data from the Input channel as the Host, or Host DMA processor, writes to the FIFO. The Host can write 40 bytes to the FIFO Input channels in the time it takes for the internal DMA processor to read 20 more bytes from it. This will keep both the Host and internal DMA operating at their maximum rates without forcing one to wait for the other.

Two methods of managing the FIFO size are possible; fixed and variable channel size. A fixed channel size is one where the channel is configured at initialization and remains unchanged throughout program execution. In a variable FIFO channel size, the configuration is changed dynamically to meet the data transmission requirements as needed. An example of a variable channel size application is the mass storage subsystem described earlier. To meet the demands of a large data block transfer the FIFO size could be fully allocated to the Input or Output channel as needed. The Thresholds are also reprogrammed to match the respective data transfer rates.

An example of a fixed channel size application might be one which uses the UPI-452 to directly control a series of stepper motors. The UPI-452 manages the motor operation and status as required. This would include pulse train, acceleration, deceleration and feedback. The Host transmits motor commands to the UPI-452 in blocks of 6–10 bytes. Each block of motor command data is preceded by a command to the UPI-452 which selects a specific motor. The UPI-452 transmits blocks of data to the Host which provides motor and overall system status. The data and embedded commands structure, indicating the specific motor, is the same. In

this example the default 64 bytes per channel might be adequate for both channels.

INTERRUPT RESPONSE TIMING

Interrupts enable the Host UPI-452 FIFO buffer interface and the internal CPU FIFO buffer interface to operate with a minimum of overhead on the respective CPU. Each CPU is "interrupted" to service the FIFO on an as needed basis only. In configuring the FIFO buffer Thresholds and choosing the appropriate internal DMA Mode the user must take into account the interrupt response time for both CPUs. These response times will affect the DMA transfer rates for each channel. By choosing FIFO channel Thresholds which reflect both the respective DMA transfer rate and the interrupt response time the user will achieve the maximum data throughput and system bus decoupling. This in turn will mean the overall available system bus bandwidth will increase.

The following section describes the FIFO interrupt interface to the Host and internal CPU. It also describes an analysis of sample interrupt response times for the Host and UPI-452 internal CPU. These equations and the example times shown are then used in the DMA section to further analyze an optimum Host UPI-452 interface.

HOST

Interrupts to the Host processor are supported by the three UPI-452 output pins; INTRQ, DRQIN/INTRQIN and DRQOUT/INTRQOUT. INTRQ is a general purpose Request For Service interrupt output. DRQIN/INTRQIN and DRQOUT/INTRQOUT pins are multiplexed to provide two special "Request for Service" FIFO interrupt request lines when DMA is disabled. A FIFO Input or Output channel Request for Service interrupt is generated based upon the value programmed in the respective channel's Threshold SFRs; Input Threshold (ITHR), and Output Threshold

(OTHR) SFRs. Additional interrupts are provided for FIFO Underrun and Overrun Errors, Data Stream Commands, and Immediate Commands. Table 4 lists the eight UPI-452 interrupt sources as they appear in the HSTAT SFR to the Host processor.

Table 4. UPI-452 to Host Interrupt Sources

HSTAT SFR Bit	Interrupt Source
HST7	Output FIFO Underrun Error
HST6	Immediate Command Out SFR Status
HST5	Data Stream Command/Data at Output FIFO Status
HST4	Output FIFO Request for Service Status
HST3	Input FIFO Overrun Error Condition
HST2	Immediate Command In SFR Status
HST1	FIFO DMA Freeze/Normal Mode Status
HST0	Input FIFO Request for Service

The interrupt response time required by the Host processor is application and system specific. In general, a typical sequence of Host interrupt response events and the approximate times associated with each are listed in Equation 1.

The example assumes the hardware configuration shown in Figure 1, iAPX 286/UPI-452 Block Diagram, with an 8259A Programmable Interrupt Controller. The timing analysis in Equation 1 also assumes the following; no other interrupt is either in process or pending, nor is the 286 in a LOCK condition. The current instruction completion time is 8 clock cycles (800 ns @ 10 MHz), or 4 bus cycles. The interrupt service routine first executes a PUSH instruction, PUSH All General Registers, to save all iAPX 286 internal registers. This requires 19 clocks (or 2.0 μ s @ 10 MHz), or 10 bus cycles (rounded to complete bus cycle). The next service routine instruction reads the UPI-452 Host Status SFR to determine the interrupt source.

It is important to note that any UPI-452 INTRQ interrupt service routine should ALWAYS mask for the Freeze Mode bit first. This will insure that Freeze Mode always has the highest priority. This will also save the time required to mask for bits which are forced inactive during Freeze Mode, before checking the Freeze Mode bit. Access to the FIFO channels by the Host is inhibited during Freeze Mode. Freeze Mode is covered in more detail below.

To initiate the interrupt the UPI-452 activates the INTRQ output. The interrupt acknowledge sequence requires two bus cycles, 400 ns (10 MHz iAPX 286), for the two INTA pulse sequence.

Equation 1. Host Interrupt Response Time

Action	Time	Bus Cycles*
Current instruction execution completion	800 ns	4
INTA sequence	400 ns	2
Interrupt service routine (time to host first READ of UPI-452)	2000 ns	10
Total Interrupt Response Time	2.3 μ s	16

NOTE:

10 MHz iAPX 286 bus cycle, 200 ns each

UPI-452 Internal

The internal CPU FIFO interrupt interface is essentially identical to that of the Host to the FIFO. Three internal interrupt sources support the FIFO operation; FIFO-Slave bus Interface Buffer, DMA Channel 0 and DMA Channel 1 Requests. These interrupts provide a maximum decoupling of the FIFO buffer and the internal CPU. The four different internal DMA Modes available add flexibility to the interface.

The FIFO-Slave Bus Interface interrupt response is also similar to the Host response to an INTRQ Request for Service interrupt. The internal CPU responds to the interrupt by reading the Slave Status (SSTAT) SFR to determine the source of the interrupt. This allows the user to prioritize the Slave Status flag response to meet the users application needs.

The internal interrupt response time is dependent on the current instruction execution, whether the interrupt is enabled, and the interrupt priority. In general, to finish execution of the current instruction, respond to the interrupt request, push the Program Counter (PC) and vector to the first instruction of the interrupt service routine requires from 38 to 86 oscillator periods (2.38 to 5.38 μ s @ 16 MHz). If the interrupt is due to an Immediate Command or DSC, additional time is required to read the Immediate Command or DSC SFR and vector to the appropriate service routine. This means two service routines back to back. One service routine to read the Slave Status (SSTAT) SFR to determine the source of the Request for Service interrupt, and second the service routine pointed to by the Immediate Command or DSC byte read from the respective SFR.

DMA is the fastest and most efficient way for the Host or internal CPU to communicate with the FIFO buffer. The UPI-452 provides support for both of these DMA paths. The two DMA paths and operations are fully independent of each other and can function simultaneously. While the Host DMA processor is performing a DMA transfer to or from the FIFO, the UPI-452 internal DMA processor can be doing the same.

Below are descriptions of both the Host and internal DMA operations. Both DMA paths can operate asynchronously and at different transfer rates. In order to make the most of this simultaneous asynchronous operation it is necessary to calculate the two transfer rates and accurately match their operations. Matching the different transfer rates is done by a combination of accurately programmed FIFO channel size and channel Thresholds. This provides the maximum Host and internal CPU to FIFO buffer interface decoupling. Below is a description of each of the two DMA operations and sample calculations for determining transfer rates. The next section of this application note, "Interface Latency", details the considerations involved in analyzing effective transfer rates when the overhead associated with each transfer is considered.

HOST FIFO DMA

DMA transfers between the Host and UPI-452 FIFO buffer are controlled by the Host CPU's DMA controller, and is independent of the UPI-452's internal two channel DMA processor. The UPI-452's internal DMA processor supports data transfers between the UPI-452 internal RAM, external RAM (via the Local Expansion Bus) and the various Special Function Registers including the FIFO Input and Output channel SFRs.

The maximum DMA transfer rate is achieved by the minimum DMA transfer cycle time to accomplish a source to destination move. The minimum Host UPI-452 FIFO DMA cycle time possible is determined by the READ and WRITE pulse widths, UPI-452 command recovery times in relation to the DMA transfer timing and DMA controller transfer mode used. Table 5 shows the relationship between the iAPX-286, iAPX-186 and UPI-452 for various DMA as well as non-DMA byte by byte transfer modes versus processor frequencies.

Host processor speed vs wait states required with UPI-452 running at 16 MHz:

Table 5. Host UPI-452
Data Transfer Performance

Processor & Speed	Wait States: Back to Back READ/WRITE's	DMA: Single Cycle	Two Cycle
iAPX-186* 8 MHz	0	N/A*	0
10 MHz	0	N/A*	0
12.5 MHz	1	N/A*	0
iAPX-286** 6 MHz	0	0	0
8 MHz	1	1	0
10 MHz	2	2	0

NOTES:

- * iAPX 186 On-chip DMA processor is two cycle operation only.
- ** iAPX 286 assumes 82258 ADMA (or other DMA) running 286 bus cycles at 286 clock rate.

In this application note system example, shown in Figure 1, DMA operation is assumed to be two bus cycle I/O to memory or memory to I/O. Two cycle DMA consists of a fetch bus cycle from the source and a store bus cycle to the destination. The data is stored in the DMA controller's registers before being sent to the destination. Single cycle DMA transfers involve a simultaneous fetch from the source and store to the destination. As the most common method of I/O-memory DMA operation, two cycle DMA transfers are the focus of this application note analysis. Equation 2 illustrates a calculation of the overall transfer rate between the FIFO buffer and external Host for a maximum FIFO size transfer. The equation does not account for the latency of initiating the DMA transfer.

Equation 2. Host FIFO DMA Transfer Rate—Input or Output Channel

$$\begin{aligned}
 &2 \text{ Cycle DMA Transfer-I/O (UPI-452) to System Memory} \\
 &= \text{FIFO channel size}^* (\text{DMA READ/WRITE FIFO time} + \text{DMA WRITE/READ Memory Time}) \\
 &= 128 \text{ bytes}^* (200 \text{ ns} + 200 \text{ ns}) \\
 &= 51.2 \mu\text{s} \\
 &= 256 \text{ bus cycles}^*
 \end{aligned}$$

NOTES:

- *10 MHz iAPX 286, 200 ns bus cycles.

The UPI-452 design is optimized for high speed DMA transfers between the Host and the FIFO buffer. The

UPI-452 internal FIFO buffer control logic provides the necessary synchronization of the external Host event and the internal CPU machine cycle during UPI-452 RD#/WR# accesses. This internal synchronization is addressed by the TCC AC specification of the UPI-452 shown in Figure 6. TCC is the time from the leading or trailing edge of a UPI-452 RD#/WR# to the same edge of the next UPI-452 RD#/WR#. The TCC time is effectively another way of measuring the system bus cycle time with reference to UPI-452 accesses.

In the iAPX-286 10 MHz system depicted in this application note the bus cycle time is 200 ns. Alternate cycle accesses of the UPI-452 during two cycle DMA operation yields a TCC time of 400 ns which is more than the TCC minimum time of 375 ns. Back to back Host UPI-452 READ/WRITE accesses may require wait states as shown in Table 5. The difference between 10 MHz iAPX-186 and 10 MHz iAPX-286 required wait states is due to the number of clock cycles in the respective bus cycle timings. The four clocks in a 10 MHz iAPX-186 bus cycle means a minimum TCC time of 400 ns versus 200 ns for a 10 MHz iAPX-286 with two clock cycle zero wait state bus cycle.

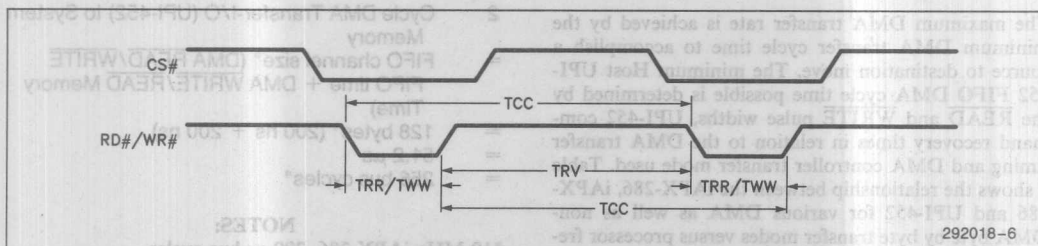
DMA handshaking between the Host DMA controller and the UPI-452 is supported by three pins on the UPI-452: DRQIN/INTRQIN, DRQOUT/INTRQOUT and DACK. The DRQIN/INTRQIN and DRQOUT/INTRQOUT outputs are two multiplexed DMA or interrupt request pins. The function of these pins is controlled by MODE SFR bit 6 (MD6). DRQIN and DRQOUT provide a direct interface to the Host system DMA controller (see Figure 1). In response to a DRQIN or DRQOUT request, the Host DMA controller initiates control of the system bus using HLD/HLDA. The FIFO Input or Output channel transfer is accomplished with a minimum of Host overhead and system bus bandwidth.

The third handshake signal pin is $\overline{\text{DACK}}$ which is used as a chip select during DMA data transfers. The UPI-452 Host READ and WRITE input signals select the respective Input and Output FIFO channel during DMA transfers. The CS and address lines provide DMA acknowledge for processors with onboard DMA controllers which do not generate a DACK signal.

The iAPX-286 Block I/O Instructions provide an alternative to two cycle DMA data transfers with approximately the same data rate. The String Input and Output instructions (INS & OUTS) when combined with the Repeat (REP) prefix, modifies INS and OUTS to provide a means of transferring blocks of data between I/O and Memory. The data transfer rate using REP INS/OUTS instructions is calculated in the same way as two cycle DMA transfer times. Each READ or WRITE would be 200 ns in a 10 MHz iAPX-286 system. The maximum transfer rate possible is 2.5 MBytes/second. The Block I/O FIFO data transfer calculation is the same as that shown in Equation 2 for two cycle DMA data transfers including TCC timing effects.

FIFO Data Structure and Host DMA

During a Host DMA write to the FIFO, if a DSC is to be written, the DMA transfer is stopped, the DSC is written and the DMA restarted. During a Host DMA read from the FIFO, if a DSC is loaded into the I/O Buffer Latch the DMA request, DRQOUT, will be deactivated (see Figure 2 above). The Host Status (HSTAT) SFR Data Stream Command bit is set and the INTRQ interrupt output goes active, if enabled. The Host responds to the interrupt as described above.



Symbol	Description	Var. Osc.	@16 MHz
TCC	Command Cycle Time	$6 * T_{clcl}$	375 ns min
TRV	Command Recovery Time	75	75 ns min

Figure 6. UPI-452 Command Cycle Timing

Once INTRQ is deactivated and the DSC has been read by the Host, the DMA request, $\overline{\text{DRQOUT}}$, is reasserted by the UPI-452. The DMA request then remains active until the transfer is complete or another DSC is loaded into the I/O Buffer Latch.

An Immediate Command written by the internal CPU during a Host DMA FIFO transfer also causes the Host Status flag and INTRQ to go active if enabled. In this case the Immediate Command would not terminate the DMA transfer unless terminated by the Host. The INTRQ line remains active until the Host reads the Host Status (HSTAT) SFR to determine the source of the interrupt.

The net effect of a Data Stream Command (DSC) on DMA data transfer rates is to add an additional factor to the data transfer rate equation. This added factor is shown in Equation 3. An Immediate Command has the same effect on the data transfer rate if the Immediate Command interrupt is recognized by the Host during a DMA transfer. If the DMA transfer is completed before the Immediate Command interrupt is recognized, the effect on the DMA transfer rate depends on whether the block being transmitted is larger than the FIFO channel size. If the block is larger than the programmed FIFO channel size the transfer rate depends on whether the Immediate Command flag or interrupt is recognized between partial block transfers.

The FIFO configuration shown in Equation 3 is arbitrary since there is no way of predicting the size relative to when a DSC would be loaded into the I/O Buffer Latch. The Host DMA rate shown is for a UPI-452

(Memory Mapped or I/O) to 286 System Memory transfer as described earlier. The equations do not account for the latency of initiating the DMA transfer.

Equation 3. Minimum host FIFO DMA Transfer Rate Including Data Stream Command(s)

$$\begin{aligned} \text{Minimum Host/FIFO DMA Transfer Rate w/ DSC} \\ &= \text{FIFO size} * \text{Host DMA 2 cycle time transfer rate} \\ &= \text{iAPX 286 interrupt response time (Eq. \#1)} \\ &= (32 \text{ bytes} * (200 \text{ ns} + 200 \text{ ns})) + 2.3 \mu\text{s} \\ &= 15.1 \mu\text{s} \\ &= 75.5 \text{ bus cycles (@10 MHz iAPX286, 200 ns bus cycle)} \end{aligned}$$

UPI-452 INTERNAL DMA PROCESSOR

The two identical internal DMA channels allow high speed data transfers from one UPI-452 writable memory space to another. The following UPI-452 memory spaces can be used with internal DMA channels:

- Internal Data Memory (RAM)
- External Data Memory (RAM)
- Special Function Registers (SFR)

The FIFO can be accessed during internal DMA operations by specifying the FIFO IN (FIN) SFR as the DMA Source Address (SAR) or the FIFO OUT (FOUT) SFR as the Destination Address (DAR). Table 6 lists the four types of internal DMA transfers and their respective timings.

Table 6. UPI-452 Internal DMA Controller Cycle Timings

Source	Destination	Machine Cycles**	@12 MHz	@16 MHz
Internal Data Mem. or SFR	Internal Data Mem. or SFR	1	1 μs	750 ns
Internal Data Mem. or SFR	External Data Mem.	1	1 μs	750 ns
Internal Data Mem.	Internal Data Mem. or SFR	1	1 μs	750 ns
*External Data Memory	External Data Memory	2	2 μs	1.5 μs

NOTES:

*External Data Memory DMA transfer applies to UPI-452 Local Bus only.

**MSC-51 Machine cycle = 12 clock cycles (TCLCL).

FIFO Data Structure and Internal DMA

The effect of Data Stream Commands and Immediate Commands on the internal DMA transfers is essentially the same as the effect on Host FIFO DMA transfers. Recognition also depends upon the programmed DMA Mode, the interrupts enabled, and their priorities. The net internal effect is the same for each possible internal case. The time required to respond to the Immediate or Data Stream Command is a function of the instruction time required. This must be calculated by the user based on the instruction cycle time given in the MSC-51 Instruction Set description in the Intel Microcontroller Handbook.

It is important to note that the internal DMA processor modes and the internal FIFO logic work together to automatically manage internal DMA transfers as data moves into and out of the FIFO. The two most appropriate internal DMA processor modes for the FIFO are FIFO Demand Mode and FIFO Alternate Cycle Mode. In FIFO Demand Mode, once the correct Slave Control and DMA Mode bits are set, the internal Input FIFO channel DMA transfer occurs whenever the Slave Control Input FIFO Request for Service flag is set. The DMA transfer continues until the flag is cleared or when the Input FIFO Read Pointer SFR (IRPR) equals zero. If data continues to be entered by the Host, the internal DMA continues until an internal interrupt of higher priority, if enabled, interrupts the DMA transfer, the internal DMA byte count reaches zero or until the Input FIFO Read Pointer equals zero. A complete description of interrupts and DMA Modes can be found in the UPI-452 Data Sheet.

DMA Modes

The internal DMA processor has four modes of operation. Each DMA channel is software programmable to operate in either Block Mode or Demand Mode. Demand Mode may be further programmed to operate in Burst or Alternate Cycle Mode. Burst Mode causes the internal processor to halt its execution and dedicate its resources exclusively to the DMA transfer. Alternate Cycle Mode causes DMA cycles and instruction cycles to occur alternately. A detailed description of each DMA Mode can be found in the UPI-452 Data Sheet.

INTERFACE LATENCY

The interface latency is the time required to accommodate all of the overhead associated with an individual data transfer. Data transfer rates between the Host system and UPI-452 FIFO, with a block size less than or equal to the programmed FIFO channel size, are calculated using the Host system DMA rate. (see Host DMA description above). In this case, the entire block could be transferred in one operation. The total latency is the time required to accomplish the block DMA transfer, the interrupt response or poll of the Host Status SFR response time, and the time required to initiate the Host DMA processor.

A DMA transfer between the Host and UPI-452 FIFO with a block size greater than the programmed FIFO channel size introduces additional overhead. This additional overhead is from three sources; first, is the time to actually perform the DMA transfer. Second, the overhead of initializing the DMA processor, third, the handshaking between each FIFO block required to transfer the entire data block. This could be time to wait for the FIFO to be emptied and/or the interrupt response time to restart the DMA transfer of the next portion of the block. A fourth component may also be the time required to respond to Underrun and Overrun FIFO Errors.

Table 7 shows six typical FIFO Input/Output channel sizes and the Host DMA transfers times for each. The timings shown reflect a 10 MHz system bus two cycle I/O to Memory DMA transfer rate of 2.5 MBytes/second as shown in Equation 1. The times given would be the same for iAPX 286 I/O block move instructions REP INS and REP OUTS as described earlier.

Table 7. Host DMA FIFO Data Transfer Times

FIFO Size:	32	43	64	85	96	128	bytes
Full or Empty	$\frac{1}{4}$	$\frac{1}{3}$	$\frac{1}{2}$	$\frac{2}{3}$	$\frac{3}{4}$	Full or Empty	
Time	12.8	17.2	25.6	34.0	38.4	51.2	μ s

Table 8 shows six typical FIFO Input/Output channel sizes and the internal DMA processor data transfers times for each. The timings shown are for a UPI-452 single cycle Burst Mode transfer at 16 MHz or 750 ns per machine cycle in or out of the FIFO channels. The

machine cycle time is that of the MSC-51 CPU; 6 states, 2 XTAL2 clock cycles each or 12 clock cycles per machine cycle. Details on the MSC-51 machine cycle timings and operation may be found in the Intel Microcontroller Handbook.

Table 8. UPI-452 Internal DMA FIFO Data Transfer Times

FIFO Size:	32	43	64	85	96	128	bytes
Full or Empty	$\frac{1}{4}$	$\frac{1}{3}$	$\frac{1}{2}$	$\frac{2}{3}$	$\frac{3}{4}$	Full or Empty	
Time	24.0	32.3	48.0	64.6	72.0	96.0	μ s

A larger than programmed FIFO channel size data block internal DMA transfer requires internal arbitration. The UPI-452 provides a variety of features which support arbitration between the two internal DMA channels and the FIFO. An example is the internal DMA processor FIFO Demand Mode described above. FIFO Demand Mode DMA transfers occur continuously until the Slave Status Request for Service Flag is deactivated. Demand Mode is especially useful for continuous data transfers requiring immediate attention. FIFO Alternate Cycle Mode provides for interleaving DMA transfers and instruction cycles to achieve a maximum of software flexibility. Both internal DMA channels can be used simultaneously to provide continuous transfer for both Input and Output FIFO channels.

Byte by byte transfers between the FIFO and internal CPU timing is a function of the specific instruction cycle time. Of the 111 MCS-51 instructions, 64 require 12 clock cycles, 45 require 24 clock cycles and 2 require 48 clock cycles. Most instructions involving SFRs are 24 clock cycles except accumulator (for example, MOV direct, A) or logical operations (ANL direct, A). Typical instruction and their timings are shown in Table 9.

Oscillator Period: @ 12 MHz = 83.3 ns

@ 16 MHz = 62.5 ns

Table 9. Typical Instruction Cycle Timings

Instruction	Oscillator Periods	@ 12 MHz	@ 16 MHz
MOV direct [†] , A	12	1 μ s	750 ns
MOV direct, direct	24	2 μ s	1.5 μ s

NOTE:

[†] Direct = 8-bit internal data locations address. This could be an Internal Data RAM location (0-255) or a SFR [i.e., I/O port, control register, etc.]

Byte by byte FIFO data transfers introduce an additional overhead factor not found in internal DMA operations. This factor is the FIFO block size to be transferred; the number of empty locations in the Output channel, or the number of bytes in the Input FIFO

channel. As described above in the FIFO Data Structure section, the block size would have to be determined by reading the channel read and write pointer and calculating the space available. Another alternative uses the FIFO Overrun and Underrun Error flags to manage the transfers by accepting error flags. In either case the instructions needed have a significant impact on the internal FIFO data transfer rate latency equation.

A typical effective internal FIFO channel transfer rate using internal DMA is shown in Equation 4. Equation 5 shows the latency using byte by byte transfers with an arbitrary factor added for internal CPU block size calculation. These two equations contrast the effective transfer rates when using internal DMA versus individual instructions to transfer 128 bytes. The effective transfer rate is approximately four times as fast using DMA versus using individual instructions (96 μ s with DMA versus 492 μ s non-DMA).

Equation 4. Effective Internal FIFO Transfer Time Using Internal DMA

Effective Internal FIFO Transfer Rate with DMA
 = FIFO channel size * Internal DMA Burst Mode Single Cycle DMA Time
 = 128 Bytes * 750 ns
 = 96 μ s

Equation 5. Effective FIFO Transfer Time Using Individual Instructions

Effective Internal FIFO Transfer Rate without DMA
 = FIFO channel size * Instruction Cycle Time + Block size calculation Time
 = 128 Bytes * (24 oscillator periods @ 16 MHz) + 20 instructions (24 oscillator period each @ 16 MHz)
 = 128 * 1.5 μ s + 300 μ s
 = 492 μ s

FIFO DMA FREEZE MODE INTERFACE

FIFO DMA Freeze Mode provides a means of locking the Host out of the FIFO Input and Output channels. FIFO DMA Freeze Mode can be invoked for a variety of reasons, for example, to reconfigure the UPI-452 Local Expansion Bus, or change the baud rate on the serial channel. The primary reason the FIFO DMA Freeze Mode is provided is to ensure that the Host does not read from or write to the FIFO while the FIFO interface is being altered. ONLY the internal CPU has the capability of altering the FIFO Special Function Registers, and these SFRs can ONLY be altered during FIFO DMA Freeze Mode. FIFO DMA Freeze Mode inhibits Host access of the FIFO while the internal CPU reconfigures the FIFO.

FIFO DMA Freeze Mode should not be arbitrarily invoked while the UPI-452 is in normal operation. Because the external CPU runs asynchronously to the internal CPU, invoking freeze mode without first properly resolving the FIFO Host interface may have serious consequences. Freeze Mode may be invoked only by the internal CPU.

The internal CPU invokes Freeze Mode by setting bit 3 of the Slave Control SFR (SC3). This automatically forces the Slave and Host Status SFR FIFO DMA Freeze Mode to In Progress (SSTAT SST5 = 0, HSTAT SFR HST1 = 1). INTRQ goes active, if enabled by MODE SFR bit 4, whenever FIFO DMA Freeze Mode is invoked to notify the Host. The Host reads the Host Status SFR to determine the source of the interrupt. INTRQ and the Slave and Host Status FIFO DMA Freeze Mode bits are reset by the Host READ of the Host Status SFR.

During FIFO DMA Freeze Mode the Host has access to the Host Status and Control SFRs. All other Host FIFO interface access is inhibited. Table 10 lists the FIFO DMA Freeze Mode status of all slave bus interface Special Function Registers. The internal DMA processor is disabled during FIFO DMA Freeze Mode and the internal CPU has write access to all of the FIFO control SFRs (Table 11).

If FIFO DMA Freeze Mode is invoked without stopping the host, only the last two bytes of data written into or read from the FIFO will be valid. The timing diagram for disabling the FIFO module to the external Host interface is illustrated in Figure 7. Due to this synchronization sequence, the UPI-452 might not go into FIFO DMA Freeze Mode immediately after the Slave Control SFR FIFO 7 DMA Freeze Mode bit (SC3) is set = 0. A special bit in the Slave Status SFR (SST5) is provided to indicate the status of the FIFO DMA Freeze Mode. The FIFO DMA Freeze Mode

operations described in this section are only valid after SST5 is cleared.

Either the Host or internal CPU can request FIFO DMA Freeze Mode. The first step is to issue an Immediate Command indicating that FIFO DMA Freeze Mode will be invoked. Upon receiving the Immediate Command, the external CPU should complete servicing all pending interrupts and DMA requests, then send an Immediate Command back to the internal CPU acknowledging the FIFO DMA Freeze Mode request. After issuing the first Immediate Command, the internal CPU should not perform any action on the FIFO until FIFO DMA Freeze Mode is invoked. The handshaking is the same in reverse if the HOST CPU initiates FIFO DMA Freeze Mode.

After the slave bus interface is frozen, the internal CPU can perform the operations listed below on the FIFO Special Function Registers. These operations are allowed only during FIFO DMA Freeze Mode. Table 11 summarizes the characteristics of all the FIFO Special Function Registers during Normal and FIFO DMA Freeze Modes.

For FIFO Reconfiguration

1. Changing the Channel Boundary Pointer SFR.
2. Changing the Input and Output Threshold SFR.

To Enhance the testability

3. Writing to the read and write pointers of the Input and Output FIFO's.
4. Writing to and reading the Host Control SFRs.
5. Controlling some bits of Host and Slave Status SFRs.
6. Reading the Immediate Command Out SFR and Writing to the Immediate Command in SFR.

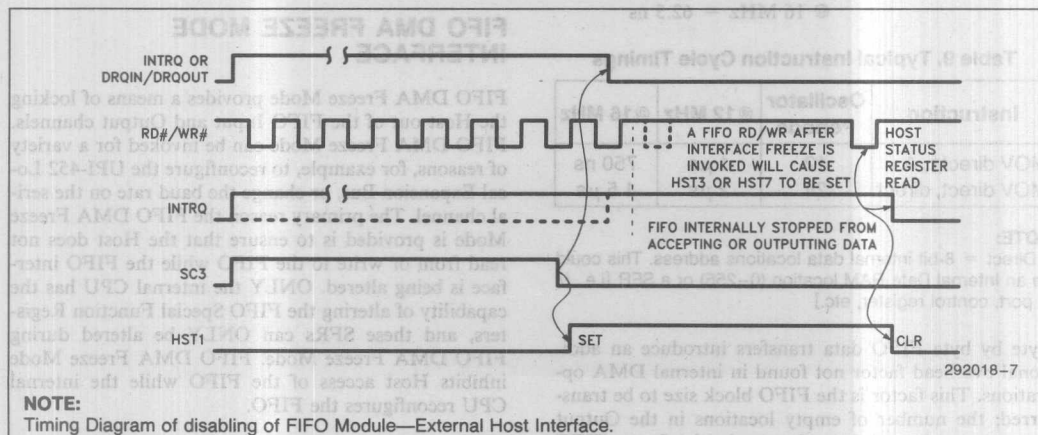


Figure 7. Disabling FIFO to Host Slave Interface Timing Diagram

The sequence of events for invoking FIFO DMA Freeze Mode are listed in Figure 8.

1. Immediate Command to request FIFO DMA Freeze Mode (interrupt)
2. Host/internal CPU interrupt response/service
3. Host/internal CPU clear/service all pending interrupts and FIFO data
4. Internal CPU sets Slave Control (SLCON) FIFO DMA Freeze Mode bit = 0, FIFO DMA Freeze Mode, Host Status SFR FIFO DMA Freeze Mode Status bit = 1, INTRQ active (high)
5. Host READ Host Status SFR
6. Internal CPU reconfigures FIFO SFRs
7. Internal CPU resets Slave Control (SLCON) FIFO DMA Freeze Mode bit = 1, Normal Mode, Host Status FIFO DMA Freeze Mode Status bit = 0.
8. Internal CPU issues Immediate Command to Host indicating that FIFO DMA Freeze Mode is complete
or
Host polls Host Status SFR FIFO DMA Freeze Mode bit to determine end of reconfiguration

Figure 8. Sequence of Events to Invoke FIFO DMA Freeze Mode

EXAMPLE CONFIGURATION

An example of the time required to reconfigure the FIFO 180 degrees, for example from 128 bytes Input to 128 bytes Output, is shown in Figure 9. The example approximates the time based on several assumptions;

1. The FIFO Input channel is full-128 bytes of data
2. Output FIFO channel is empty-1 byte
3. No Data Stream Commands in the FIFO.

4. The Immediate Command interrupt is responded to immediately—highest priority—by Host and internal CPU.
5. Respective interrupt response times
 - a. Host (Equation 3 above) = approximately 1.6 μ s
 - b. Internal CPU is 86 oscillator periods or approximately 5.38 μ s worst case.

Event	Time
Immediate Command from Host to UPI-452 to request FIFO DMA Freeze Mode (iAPX286 WRITE)	0.30 μ s
Internal CPU interrupt response/service	5.38 μ s
Internal CPU clears FIFO-128 bytes DMA	96.00 μ s
Internal CPU sets Slave Control Freeze Mode bit	0.75 μ s
Immediate Command to Host-Freeze Mode in progress Host Immediate Command interrupt response	2.3 μ s
Internal CPU reconfigures FIFO SFRs	
Channel Boundary Pointer SFR	0.75 μ s
Input Threshold SFR	0.75 μ s
Output Threshold SFR	0.75 μ s
Internal CPU resets Slave Control (SLCON) Freeze Mode bit = 1, Normal Mode, and automatically resets Host Status FIFO DMA Freeze Mode bit	2.3 μ s
Internal CPU writes Immediate Command Out	0.75 μ s
Host Immediate Command interrupt service	2.3 μ s
Total Minimum Time to Reconfigure FIFO	112.33 μ s

Figure 9. Sequence of Events to Invoke FIFO DMA Freeze Mode and Timings

Table 10. Slave Bus Interface Status During FIFO DMA Freezer Mode

		Interface Pins;							Operation In	Status In
DACK	CS	A2	A1	A0	READ	WRITE			Normal Mode	Freeze Mode
1	0	0	1	0	0	1			Read Host Status SFR	Operational
1	0	0	1	1	0	1			Read Host Control SFR	Operational
1	0	0	1	1	1	0			Write Host Control SFR	Disabled
1	0	0	0	0	0	1			Data or DMA data from Output Channel	Disabled
1	0	0	0	0	1	0			Data or DMA data to Input Channel	Disabled
1	0	0	0	1	0	1			Data Stream Command from Output Channel	Disabled
1	0	0	0	1	1	0			Data Stream Command to Input Channel	Disabled
1	0	1	0	0	0	1			Read Immediate Command Out from Output Channel	Disabled
1	0	1	0	0	1	0			Write Immediate Command In to Input Channel	Disabled
0	X	X	X	X	0	1			DMA Data from Output Channel	Disabled
0	X	X	X	X	1	0			DMA Data to Input Channel	Disabled

NOTE:
X = don't care

Table 11. FIFO SFR's Characteristics During FIFO DMA Freeze Mode

Label	Name	Normal Operation (SST5 = 1)	Freeze Mode Operation (SST5 = 0)
HCON	Host Control	Not Accessible	Read & Write
HSTAT	Host Status	Read Only	Read & Write
SLCON	Slave Control	Read & Write	Read & Write
SSTAT	Slave Status	Read Only	Read & Write
IEP	Interrupt Enable & Priority	Read & Write	Read & Write
MODE	Mode Register	Read & Write	Read & Write
IWPR	Input FIFO Write Pointer	Read Only	Read & Write
IRPR	Input FIFO Read Pointer	Read Only	Read & Write
OWPR	Output FIFO Write Pointer	Read Only	Read & Write
ORPR	Output FIFO Read Pointer	Read Only	Read & Write
CBP	Channel Boundary Pointer	Read Only	Read & Write
IMIN	Immediate Command In	Read Only	Read & Write
IMONT	Immediate Command Out	Read & Write	Read & Write
FIN	FIFO IN	Read Only	Read Only
CIN	COMMAND IN	Read Only	Read Only
FOUT	FIFO OUT	Read & Write	Read & Write
COUT	COMMAND OUT	Read & Write	Read & Write
ITHR	Input FIFO Threshold	Read Only	Read & Write
OTHR	Other FIFO Threshold	Read Only	Read & Write



APPLICATION NOTE

AP-283

Since the reader is familiar with the 8044 data sheet and the SDLC communications protocol.

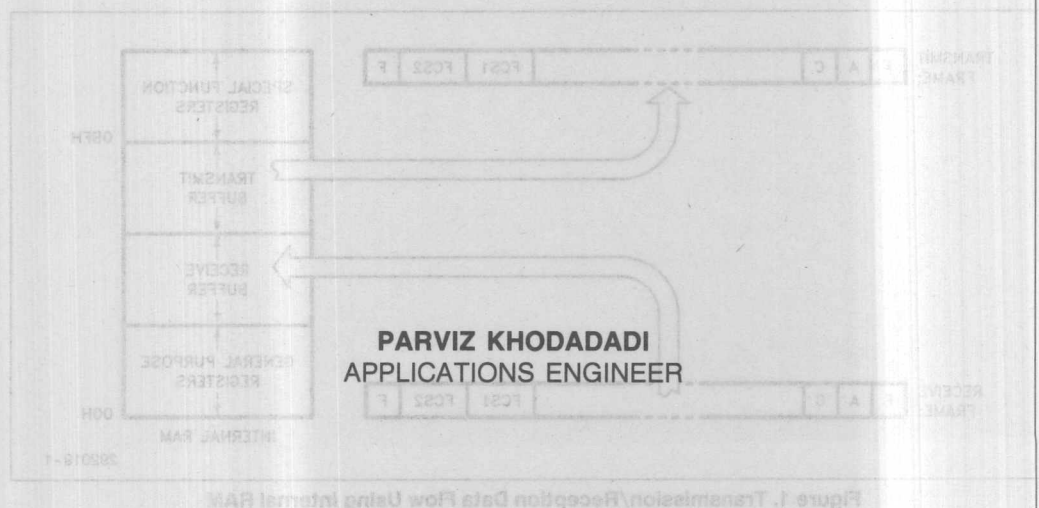
1.1 Normal Operation

In Normal operation the on-chip CPU and the SIU operate in parallel. The SIU handles the serial communication task while the CPU processes the contents of the on-chip transmit and receive buffer, services interrupt routines, or performs the local real time processing tasks.

The 192 bytes of on-chip RAM serves as the interface buffer between the CPU and the SIU, used by both as a receive and transmit buffer. Some of the internal RAM space is used as general purpose registers (e.g. R0-R7).

The remaining bytes may be divided into at least two sections: one section for the transmit buffer and the other section for the receive buffer. In some applications, the 192 byte internal RAM size imposes a limitation on the size of the information field of each frame and, consequently, achieves less than optimal information throughput.

Figure 1 illustrates the flow of data when internal RAM is used as the receive and transmit buffer. The on-chip CPU allocates a receive buffer in the internal RAM and enables the SIU. A receiving SDLC frame is processed by the SIU and information bytes are loaded into the internal RAM. For transmission, the CPU loads information bytes into the internal RAM and enables the SIU. The SIU transmits the information bytes in SDLC format.



PARVIZ KHODADADI
APPLICATIONS ENGINEER

1.0 INTRODUCTION

The 8044 is a serial communication microcontroller known as the RUPI (Remote Universal Peripheral Interface). It merges the popular 8051 8-bit microcontroller with an intelligent, high performance HDLC/SDLC serial communication controller called the Serial Interface Unit (SIU). The chip provides all features of the microcontroller and supports the Synchronous Data Link Control (SDLC) communications protocol.

There are two methods of operation relating to frame size:

- 1) Normal operation (limited frame size)
- 2) Expanded operation (unlimited frame size)

In Normal operation the internal 192 byte RAM is used as the receive and transmit buffer. In this operation, the chip supports data rates up to 2.4 Mbps externally clocked and 375 Kbps self-clocked. For frame sizes greater than 192 bytes, Expanded operation is required. In Expanded operation the external RAM, in conjunction with the internal RAM, is used as the transmit and receive buffer. In this operation, the chip supports data rates up to 500 Kbps externally clocked and 375 Kbps self-clocked. In both cases, the SIU handles many of the data link functions in hardware, and the chip can be configured in either Auto or Flexible mode.

The discussion that follows describes the operation of the chip and the behavior of the serial interface unit. Both Normal and Expanded operations will be further explained with extra emphasis on Expanded operation and its supporting software. Two examples of SDLC communication systems will also be covered, where the chip is used in Expanded operation. The discussion as-

sumes that the reader is familiar with the 8044 data sheet and the SDLC communications protocol.

1.1 Normal Operation

In Normal operation the on-chip CPU and the SIU operate in parallel. The SIU handles the serial communication task while the CPU processes the contents of the on-chip transmit and receiver buffer, services interrupt routines, or performs the local real time processing tasks.

The 192 bytes of on-chip RAM serves as the interface buffer between the CPU and the SIU, used by both as a receive and transmit buffer. Some of the internal RAM space is used as general purpose registers (e.g. R0-R7). The remaining bytes may be divided into at least two sections: one section for the transmit buffer and the other section for the receive buffer. In some applications, the 192 byte internal RAM size imposes a limitation on the size of the information field of each frame and, consequently, achieves less than optimal information throughput.

Figure 1 illustrates the flow of data when internal RAM is used as the receive and transmit buffer. The on-chip CPU allocates a receive buffer in the internal RAM and enables the SIU. A receiving SDLC frame is processed by the SIU and the information bytes of the frame, if any, are stored in the internal RAM. Then, the SIU informs the CPU of the received bytes (Serial Channel interrupt). For transmission, the CPU loads the transmitting bytes into the internal RAM and enables the SIU. The SIU transmits the information bytes in SDLC format.

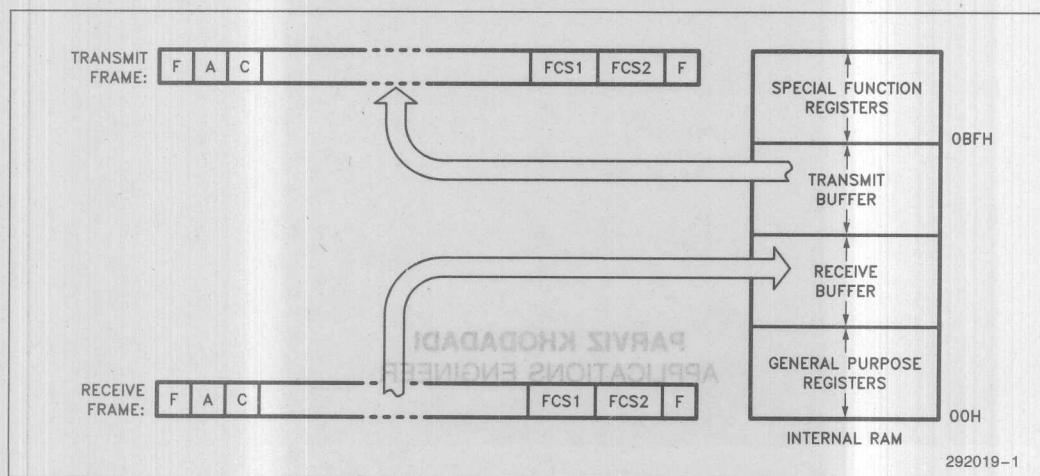


Figure 1. Transmission/Reception Data Flow Using Internal RAM

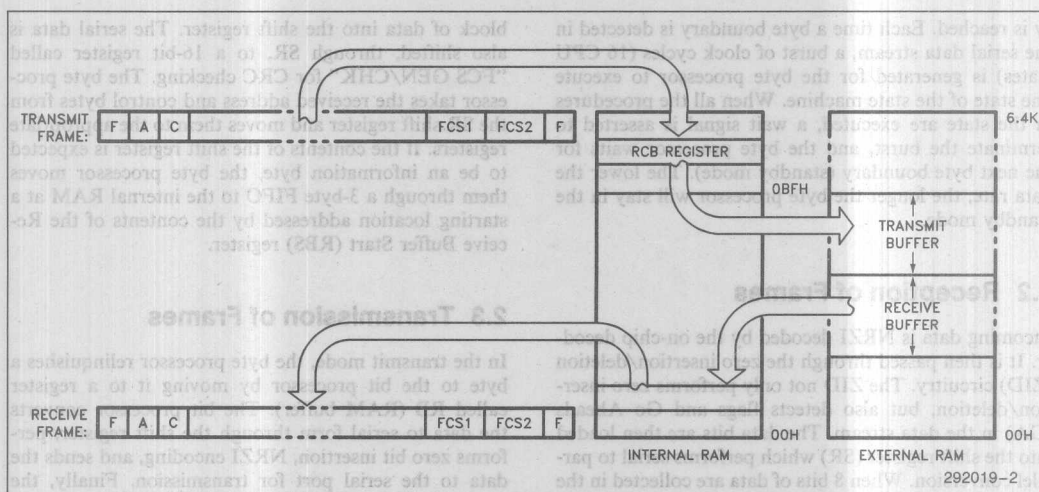


Figure 2. Transmission/Reception Data Flow Using External RAM

1.2 Expanded Operation

In Expanded operation the on-chip CPU monitors the state of the SIU, and moves data from/to external buffer to/from the internal RAM and registers while reception/transmission is taking place. If the CPU must service an interrupt during transmission or reception of a frame or transmit from internal RAM, the chip can shift to Normal operation.

There is a special function register called SIUST, the contents of which dictate the operation of the SIU. Also, at data rates lower than 2.4 Mbps, one section of the SIU, in fixed intervals during transmission and reception, is in the "standby" mode and performs no function. The above two characteristics make it possible to program the CPU to move data to/from external RAM and to force the SIU to repeat or skip some desired hardware tasks while transmission or reception is taking place. With these modifications, external RAM can be utilized as a transmit and receive buffer instead of the internal RAM.

Figure 2 graphically shows the flow of data when external RAM is used. For reception, the receiving bytes are loaded into the Receive Control Byte (RCB) register. Then, the data in RCB is moved to external RAM and the SIU is forced to load the next byte into the RCB register - The chip believes it is receiving a control byte continuously. For transmission, Information bytes (I-bytes) are loaded into a location in the internal RAM and the chip is forced to transmit the contents of this location repeatedly.

Discussion of expanded operation is continued in sections 4 and 5. First, however, sections 2 and 3 describe

features of the 8044 which are necessary to further explain expanded operation.

2.0 THE SERIAL INTERFACE UNIT

2.1 Hardware Description

The Serial Interface Unit (SIU) of the RUPI, shown in Figure 3, is divided functionally into a Bit Processor (BIP) and a Byte Processor (BYP), each sharing some common timing and control logic. The bit processor is the interface between the SIU bus and the serial port pins. It performs all functions necessary to transmit/receive a byte of data to/from the serial data line (shifting, NRZI coding, zero insertion/deletion, etc.). The byte processor manipulates bytes of data to perform message formatting, transmitting, and receiving functions. For example, moving bytes from/to the special function registers to/from the bit processor.

The byte processor is controlled by a Finite-State Machine (FSM). For every receiving/transmitting byte, the byte processor executes one state. It then jumps to the next state or repeats the same state. These states will be explained in section 3. The status of the FSM is kept in an 8-bit register called SIUST (SIU State Counter). This register is used to manipulate the behavior of the byte processor.

As the name implies, the bit processor processes data one bit at a time. The speed of the bit processor is a function of the serial channel data rate. When one byte of data is processed by the bit processor, a byte bounda-

ry is reached. Each time a byte boundary is detected in the serial data stream, a burst of clock cycles (16 CPU states) is generated for the byte processor to execute one state of the state machine. When all the procedures in the state are executed, a wait signal is asserted to terminate the burst, and the byte processor waits for the next byte boundary (standby mode). The lower the data rate, the longer the byte processor will stay in the standby mode.

2.2 Reception of Frames

Incoming data is NRZI decoded by the on-chip decoder. It is then passed through the zero insertion/deletion (ZID) circuitry. The ZID not only performs zero insertion/deletion, but also detects flags and Go Aheads (GA) in the data stream. The data bits are then loaded into the shift register (SR) which performs serial to parallel conversion. When 8 bits of data are collected in the shift register, the bit processor triggers the byte processor to process the byte, and it proceeds to load the next

block of data into the shift register. The serial data is also shifted, through SR, to a 16-bit register called "FCS GEN/CHK" for CRC checking. The byte processor takes the received address and control bytes from the SR shift register and moves them to the appropriate registers. If the contents of the shift register is expected to be an information byte, the byte processor moves them through a 3-byte FIFO to the internal RAM at a starting location addressed by the contents of the Receive Buffer Start (RBS) register.

2.3 Transmission of Frames

In the transmit mode, the byte processor relinquishes a byte to the bit processor by moving it to a register called RB (RAM buffer). The bit processor converts the data to serial form through the shift register, performs zero bit insertion, NRZI encoding, and sends the data to the serial port for transmission. Finally, the contents of the FCS GEN/CHK and the closing flag are routed to the serial port for transmission.

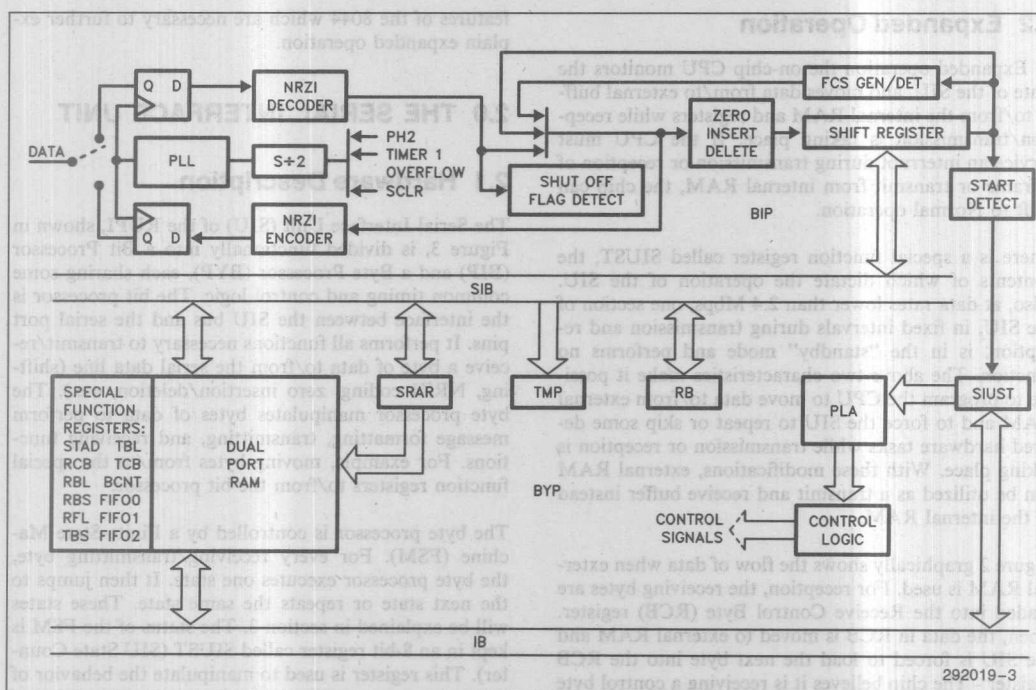


Figure 3. SIU Block Diagram

3.0 TRANSMIT AND RECEIVE STATES

The simplified receive and transmit state diagrams are shown in Figures 4 and 5, respectively. The numbers on the left of each state represent the contents of the SIUST register when the byte processor is in the standby mode, and the instructions on the right of each state represent the "state procedures" of that state. When the byte processor executes these procedures the least three significant bits of the SIUST register are being incremented while the other bits remain unchanged. The byte processor will jump from one state to another without going into the standby mode when a conditional jump procedure executed by the byte processor is true.

3.1 Receive State Sequence

When an opening flag (7EH) is detected by the bit processor, the byte processor is triggered to execute the procedures of the FLAG state. In the FLAG state, the byte processor loads the contents of the RBS register into the Special RAM (SRAR) register. SRAR is the pointer to the internal RAM. The byte processor decrements the contents of the Receive Buffer Length (RBL) register and loads them into the DMA Count (DCNT) register. The FCS GEN/CHK circuit is turned on to monitor the serial data stream for Frame Check Sequence functions as per SDLC specifications.

Assuming there is an address field in the frame, contents of the SIUST register will then be changed to 08H, causing the byte processor to jump to the ADDRESS state and wait (standby) for the next byte boundary. As soon as the bit processor moves the address byte into the SR shift register, a byte boundary is achieved and the byte processor is triggered to execute the procedures in the ADDRESS state.

In the ADDRESS state the received station address is compared to the contents of the STAD register. If there is no match, or the address is not the broadcast address (FFH), reception will be aborted (SIUST = 01H). Otherwise, the byte processor jumps to the CONTROL state (SIUST = 10H) and goes into standby mode.

The byte processor jumps to the CONTROL state if there exists a control field in the receiving frame. In this state the control byte is moved to the RCB register by the byte processor. Note that the only action taken in this state is that a received byte, processed by the bit processor, is moved to RCB. There is no other hardware task performed, and DCNT and SRAR are not affected in this state.

The next two states, PUSH-1 and PUSH-2, will be executed if Frame check sequence (NFCS = 0) option is selected. In these two states the first and second bytes

of the information field are pushed into the 3-byte FIFO (FIFO0, FIFO1, FIFO2) and the Receive Field Length register (RFL) is set to zero. The 3-byte FIFO is used as a pipeline to move received bytes into the internal RAM. The FIFO prevents transfer of CRC bytes and the closing flag to the receive buffer (i.e., when the ending flag is received, the contents of FIFO are FLAG, FCS1, and FCS0.) The three byte FIFO is collapsed to one byte in No FCS mode.

In the DMA-LOOP state the byte processor pushes a byte from SR to FIFO0, moves the contents of FIFO2 to the internal RAM addressed by the contents of SRAR, increments the SRAR and RFL registers, and decrements the DCNT register. If more information bytes are expected, the byte processor repeats this state on the next byte boundaries until DMA Buffer End occurs. The DMA Buffer End occurs if SRAR reaches 0BFH (192 decimal), DCNT reaches zero, or the RBP bit of the STS register is set.

The BOV-LOOP state, the last state, is executed if there is a buffer overrun. Buffer overrun occurs when the number of information bytes received is larger than the length of the receive buffer (RFL > RBL). This state is executed until the closing flag is received.

At the end of reception, if the FCS option is used, the closing flag and the FCS bytes will remain in the 3-byte FIFO. The contents of the RCB register are used to update the NSNR (Receive/Send Count) register. The SIU updates the STS register and sets the serial interrupt.

3.2 Transmit State Sequence

Setting the RTS bit puts the SIU in the transmit mode. When the CTS pin goes active, the byte processor goes into START-XMIT state. In this state the opening flag is moved into the RAM Buffer (RB) register. The byte processor jumps to the next state and goes into the standby mode.

If the Pre-Frame Sync (PFS) option is selected, the PFS1 and PFS2 states will be executed to transmit the two Pre-Frame Sync bytes (00H or 55H). In these two states the contents of the Pre-Frame Sync generator are sent to the serial port while the Zero Insertion Circuit (ZID) is turned off. ZID is turned back on automatically on the next byte boundary.

If the PFS option is not chosen, the byte processor jumps to the FLAG state. In this state, the byte processor moves the contents of TBS into the SRAR register, decrements TBL and moves the contents into the DCNT register. The byte processor turns off the ZID and turns on FCS GEN/CHK. The contents of FCS GEN/CHK are not transmitted unless the NFCS bit is

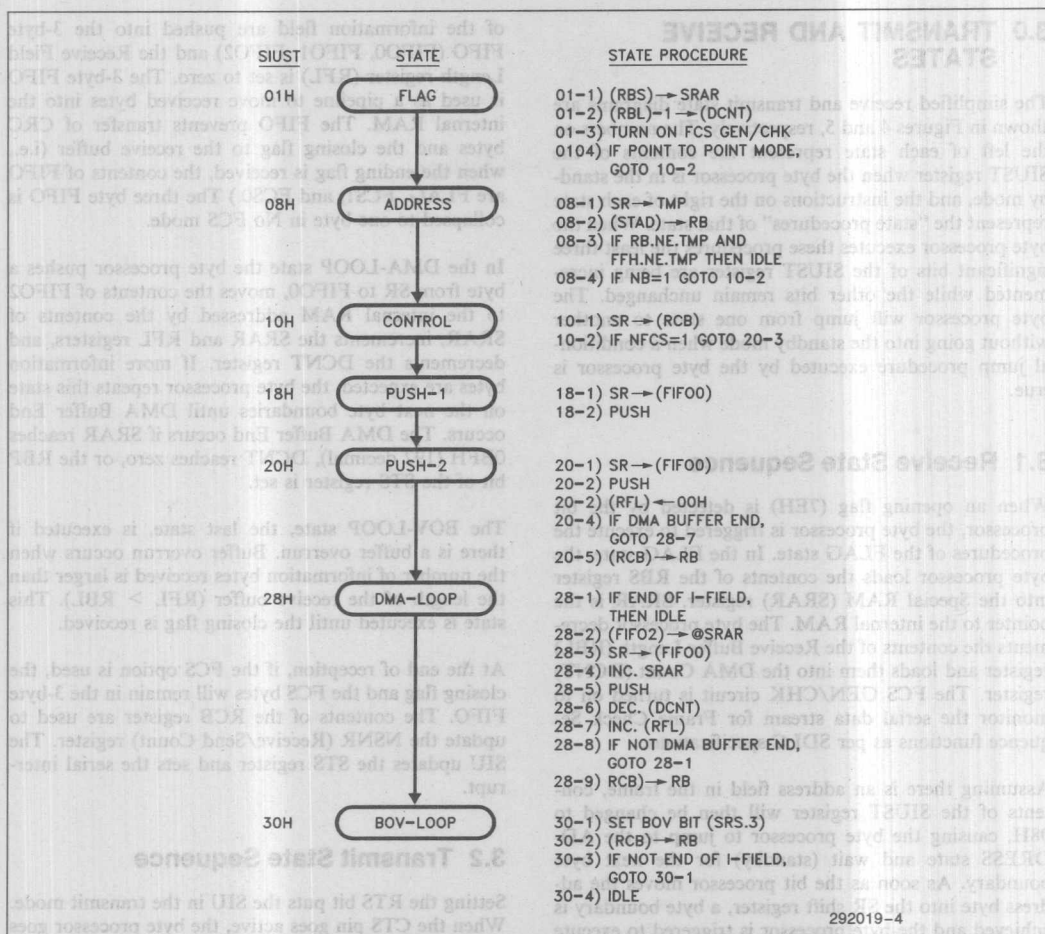


Figure 4. Receive State Diagram

set. If a frame with the address field is chosen, it moves the contents of the STAD register into the RB register for transmission. At the same time, the opening flag is being transmitted by the bit processor.

In the ADDRESS (SIUST = A0H) and CONTROL (SIUST = A8H) states, TCB and the first information byte are loaded into the RB register for transmission, respectively. Note that in the CONTROL state, none of the registers (e.g. DCNT, SRAR) are incremented, and ZID and FCS GEN/CHK are not turned on or off.

The procedures in the DMA-LOOP state are similar to the procedures of the DMA-LOOP in the receive state diagram. The SRAR register pointer to the internal RAM is incremented, and the DCNT register is decremented. The contents of DCNT reach zero when all the information bytes from the transmit buffer are transmitted. A byte from RAM is moved to the RB register for transmission. This state is executed on the following

byte boundaries until all the information bytes are transmitted.

The FCS1 and the FCS2 states are executed to transmit the Frame Check Sequence bytes generated by the FCS generator, and the END-FLAG state is executed to transmit the closing flag.

The XMIT-ACTION and the ABORT-ACTION states are executed by the byte processor to synchronize the SIU with the CPU clock. The XMIT-ACTION or the ABORT-ACTION state is repeated until the byte processor status is updated. At the end, the STS and the TMOD registers are updated.

The two ABORT-SEQUENCE states (SIUST = E0H and SIUST = E8H) are executed only if transmission is aborted by the CPU (RTS or TBF bit of the STS register is cleared) or by the serial data link (CTS signal goes inactive or shut-off occurs in loop mode.)

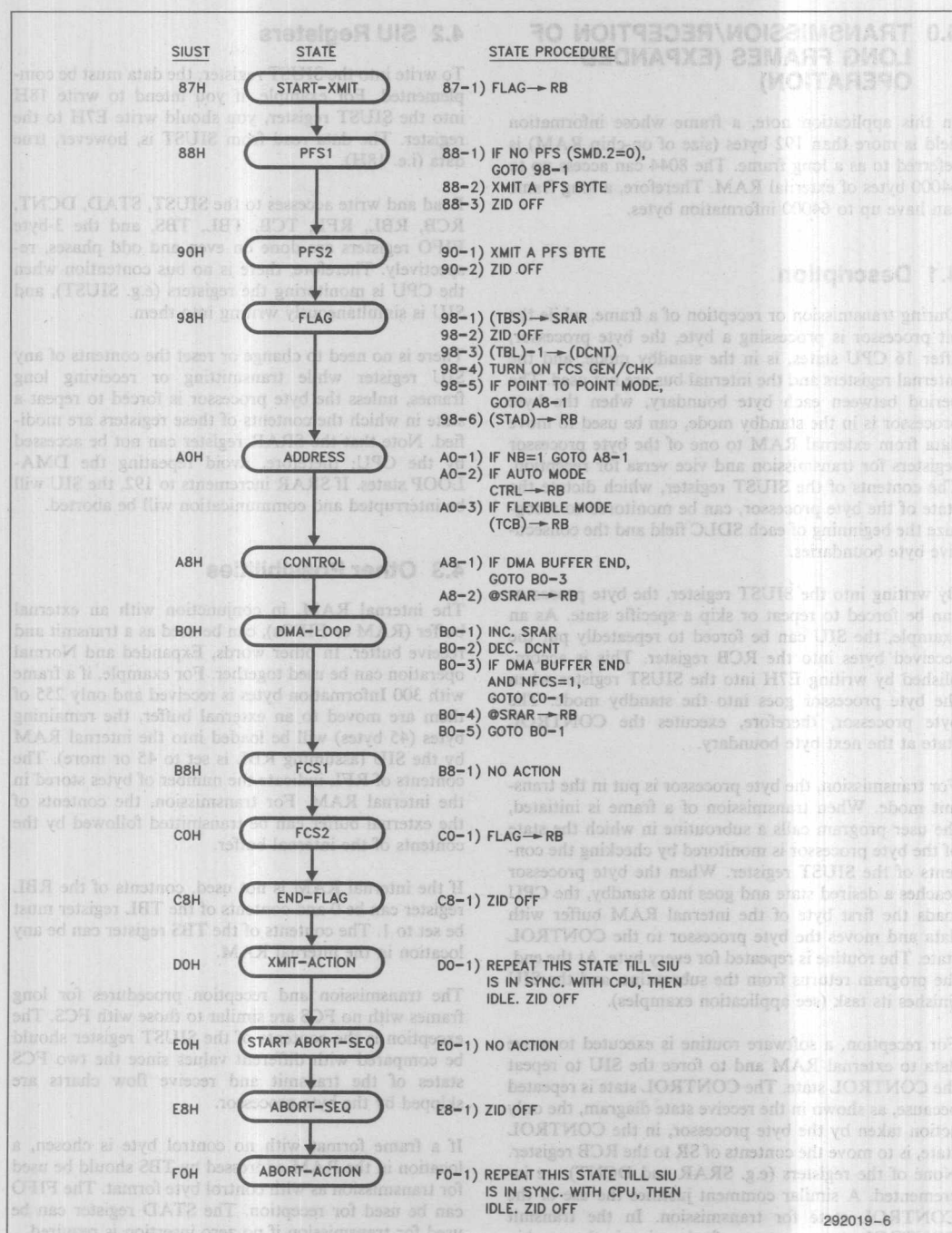


Figure 5. Transmit State Diagram

4.0 TRANSMISSION/RECEPTION OF LONG FRAMES (EXPANDED OPERATION)

In this application note, a frame whose information field is more than 192 bytes (size of on-chip RAM) is referred to as a long frame. The 8044 can access up to 64000 bytes of external RAM. Therefore, a long frame can have up to 64000 information bytes.

4.1 Description

During transmission or reception of a frame, while the bit processor is processing a byte, the byte processor, after 16 CPU states, is in the standby mode, and the internal registers and the internal bus are not used. The period between each byte boundary, when the byte processor is in the standby mode, can be used to move data from external RAM to one of the byte processor registers for transmission and vice versa for reception. The contents of the SIUST register, which dictate the state of the byte processor, can be monitored to recognize the beginning of each SDLC field and the consecutive byte boundaries.

By writing into the SIUST register, the byte processor can be forced to repeat or skip a specific state. As an example, the SIU can be forced to repeatedly put the received bytes into the RCB register. This is accomplished by writing E7H into the SIUST register when the byte processor goes into the standby mode. The byte processor, therefore, executes the CONTROL state at the next byte boundary.

For transmission, the byte processor is put in the transmit mode. When transmission of a frame is initiated, the user program calls a subroutine in which the state of the byte processor is monitored by checking the contents of the SIUST register. When the byte processor reaches a desired state and goes into standby, the CPU loads the first byte of the internal RAM buffer with data and moves the byte processor to the CONTROL state. The routine is repeated for every byte. At the end, the program returns from the subroutine, and the SIU finishes its task (see application examples).

For reception, a software routine is executed to move data to external RAM and to force the SIU to repeat the CONTROL state. The CONTROL state is repeated because, as shown in the receive state diagram, the only action taken by the byte processor, in the CONTROL state, is to move the contents of SR to the RCB register. None of the registers (e.g. SRAR and DCNT) are incremented. A similar comment justifies the use of the CONTROL state for transmission. In the transmit CONTROL state, contents of a location in the on-chip RAM addressed by TBS is moved to RB for transmission.

4.2 SIU Registers

To write into the SIUST register, the data must be complemented. For example, if you intend to write 18H into the SIUST register, you should write E7H to the register. The data read from SIUST is, however, true data (i.e. 18H).

Read and write accesses to the SIUST, STAD, DCNT, RCB, RBL, RFL, TCB, TBL, TBS, and the 3-byte FIFO registers are done on even and odd phases, respectively. Therefore, there is no bus contention when the CPU is monitoring the registers (e.g. SIUST), and SIU is simultaneously writing into them.

There is no need to change or reset the contents of any SIU register while transmitting or receiving long frames, unless the byte processor is forced to repeat a state in which the contents of these registers are modified. Note that the SRAR register can not be accessed by the CPU; therefore, avoid repeating the DMA-LOOP states. If SRAR increments to 192, the SIU will be interrupted and communication will be aborted.

4.3 Other Possibilities

The internal RAM, in conjunction with an external buffer (RAM or FIFOs), can be used as a transmit and receive buffer. In other words, Expanded and Normal operation can be used together. For example, if a frame with 300 Information bytes is received and only 255 of them are moved to an external buffer, the remaining bytes (45 bytes) will be loaded into the internal RAM by the SIU (assuming RBL is set to 45 or more). The contents of RFL indicate the number of bytes stored in the internal RAM. For transmission, the contents of the external buffer can be transmitted followed by the contents of the internal buffer.

If the internal RAM is not used, contents of the RBL register can be 0 and contents of the TBL register must be set to 1. The contents of the TBS register can be any location in the internal RAM.

The transmission and reception procedures for long frames with no FCS are similar to those with FCS. The exception is the contents of the SIUST register should be compared with different values since the two FCS states of the transmit and receive flow charts are skipped by the byte processor.

If a frame format with no control byte is chosen, a location in the RAM addressed by TBS should be used for transmission as with control byte format. The FIFO can be used for reception. The STAD register can be used for transmission if no zero insertion is required.

If the RUPI is used in Auto mode (see Section 5), it will still respond to RR, RNR, REJ, and Unnumbered Poll (UP) SDLC commands with RR or RNR automatically, without using any transmit routine. For example, if the on-chip CPU is busy performing some real time operations, the SIU can transmit an information frame from the internal buffer or transmit a supervisory frame without the help of CPU (Normal operation).

Maximum data rate using this feature is limited primarily by the number of instructions needed to be executed during the standby mode.

Transmission or reception of a frame can be timed out so that the CPU will not hang up in the transmit or receive procedures if a frame is aborted. Or, if the data rate allows enough time (standby time is long enough), the CPU can monitor the SIUST register for idle mode (SIUST = 01H).

It is also possible to transmit multiple opening or closing flags by forcing the byte processor to repeat the END-FLAG state.

4.4 Maximum Data Rate in Expanded Operation

Assuming there is no zero-insertion/deletion, the bit processor requires eight serial clock periods to process one block of data. The byte processor, running on the CPU clock, processes one byte of data in 16 CPU states (one state of the state diagrams). Each CPU state is two oscillator periods. At an oscillator frequency of 12 MHz, the CPU clock is 6 MHz, and 16 CPU states is 2.7 μ s. At a 3 Mbit rate with no zero-insertion/deletion, there is exactly enough time to execute one state per byte (16 states at 6 MHz = 8 bits at 3M baud). In other words, the standby time is zero.

Figure 6 demonstrates portions of the timing relationship between the byte processor and the bit processor. In each state, the actions taken by the processors, plus the contents of the SIUST register, are shown. When the byte processor is running, the contents of SIUST are unknown. However, when it is in the standby mode, its contents are determinable.

The maximum data rate for transmitting and receiving long frames depends on the number of instructions needed to be executed during standby, and is propor-

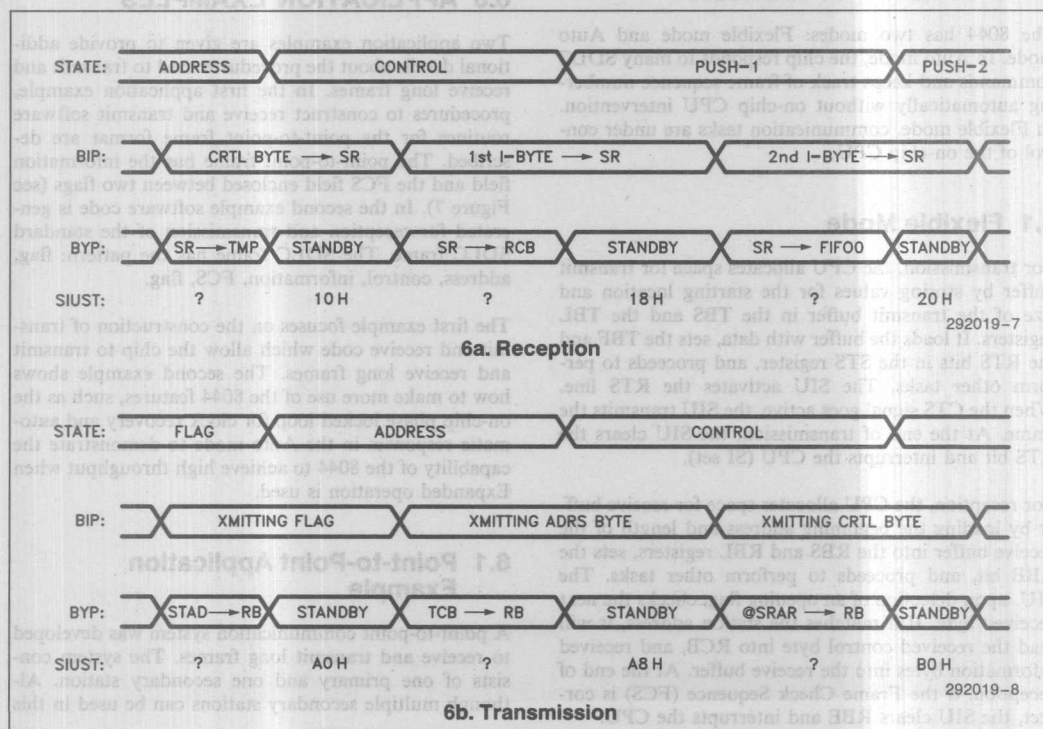


Figure 6. Portions of the BIP/BYP Timing Relationship

tional to the oscillator frequency. The time the byte processor is in the standby mode, waiting for the bit processor to deliver a processed byte, is at least equal to eight serial clock periods minus 16 CPU states. If an inserted zero is in the block of data, the bit processor will process the byte in nine serial clock periods.

The equation for theoretical maximum data rate is given as:

$$\frac{(2TCLCL) \times (16 \text{ states}) + (\# \text{ of instruction cycles}) \times (12TCLCL)}{(8TDCY)} = \text{Equation (1)}$$

Where: TCLCL is the oscillator period.
TDCY is the serial clock period.

At an oscillator frequency of 12 MHz and baud rate of 375 Kbps, about 18 instruction cycles can be executed when the byte processor is in the standby mode. At a 9600 baud rate, there is time to execute about 830 instruction cycles—plenty of time to service a long interrupt routine or perform bit-manipulation or arithmetic operations on the data while transmission or reception is taking place.

5.0 MODES OF OPERATION

The 8044 has two modes: Flexible mode and Auto mode. In Auto mode, the chip responds to many SDLC commands and keeps track of frame sequence numbering automatically without on-chip CPU intervention. In Flexible mode, communication tasks are under control of the on-chip CPU.

5.1 Flexible Mode

For transmission, the CPU allocates space for transmit buffer by storing values for the starting location and size of the transmit buffer in the TBS and the TBL registers. It loads the buffer with data, sets the TBF and the RTS bits in the STS register, and proceeds to perform other tasks. The SIU activates the RTS line. When the CTS signal goes active, the SIU transmits the frame. At the end of transmission, the SIU clears the RTS bit and interrupts the CPU (SI set).

For reception, the CPU allocates space for receive buffer by loading the beginning address and length of the receive buffer into the RBS and RBL registers, sets the RBE bit, and proceeds to perform other tasks. The SIU, upon detection of an opening flag, checks the next received byte. If it matches the station address, it will load the received control byte into RCB, and received information bytes into the receive buffer. At the end of reception, if the Frame Check Sequence (FCS) is correct, the SIU clears RBE and interrupts the CPU.

5.2 Auto Mode

In the Auto mode, the 8044 can only be a secondary station operating in the SDLC "Normal Response Mode". The 8044 in Auto mode does not transmit messages unless it is polled by the primary.

For transmission of an information frame, the CPU allocates space for the transmit buffer, loads the buffer with data, and sets the TBF bit. The SIU will transmit the frame when it receives a valid poll-frame. A frame whose poll bit of the control byte is set, is a poll-frame. The poll bit causes the RTS bit to be set. If TBF were not set, the SIU would respond with Receive Not Ready (RNR) SDLC command if RBP = 1, or with Receive Ready (RR) SDLC command if RBP = 0. After transmission RTS is cleared, and the CPU is not interrupted.

For reception, the procedure is the same as that of Flexible mode. In addition, the SIU sets the RTS bit if the received frame is a poll-frame (causing an automatic response) and increments the NS and NR counts accordingly.

6.0 APPLICATION EXAMPLES

Two application examples are given to provide additional details about the procedures used to transmit and receive long frames. In the first application example, procedures to construct receive and transmit software routines for the point-to-point frame format are described. The point-to-point frame has the information field and the FCS field enclosed between two flags (see Figure 7). In the second example software code is generated for reception and transmission of the standard SDLC frame. The SDLC frame has the pattern: flag, address, control, information, FCS, flag.

The first example focuses on the construction of transmit and receive code which allow the chip to transmit and receive long frames. The second example shows how to make more use of the 8044 features, such as the on-chip phase locked loop for clock recovery and automatic responses in the Auto mode to demonstrate the capability of the 8044 to achieve high throughput when Expanded operation is used.

6.1 Point-to-Point Application Example

A point-to-point communication system was developed to receive and transmit long frames. The system consists of one primary and one secondary station. Although multiple secondary stations can be used in this

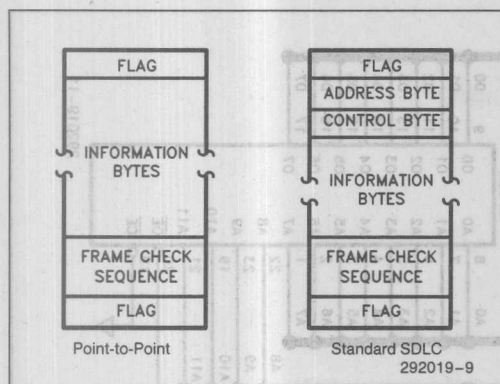


Figure 7. Point-to-Point and Standard SDLC Frame Formats

system, one secondary is chosen to simplify the primary station's software and focus on the long frame software code. Both the primary and the secondary stations are in Flexible mode and the external clock option is used for the serial channel. The maximum data rate is 500 Kbps. The FCS bytes are generated and checked automatically by both stations.

6.1.1 POLLING SEQUENCE

The polling sequence, shown in Figure 8, takes place continuously between the primary and the secondary stations. The primary transmits a frame with one information byte to the secondary. The information byte is used by the secondary as an address byte. The secondary checks the received byte, and if the address matches, the secondary responds with a long frame. In this example, the information field of the frame is chosen to be 255 bytes long. Since there is only one secondary station, the address always matches. Upon successful reception of the long frame, the primary transmits another frame to the secondary station.

6.1.2 HARDWARE

The schematic of the secondary station is given in Figure 9. The circuit of the primary station is identical to the secondary station with the exception of pin 11

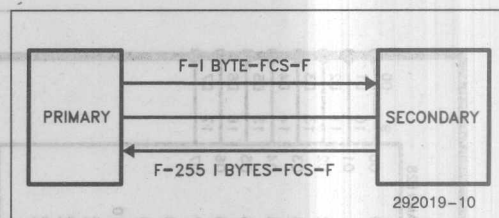


Figure 8. Secondary Responses to Primary Station Commands

(DATA) being connected to pin 14 (T0). In the primary station, the 8044 is interrupted when activity is detected on the communication line by the on-chip timer (in counter mode). This is explained more later. The serial clock to both stations is supplied by a pulse generator. The output of the pulse generator (not shown in the diagram) is connected to pin 15 of the 8044s. Since the two stations are located near each other (less than 4 feet), line drivers are not used.

The central processor of each station is the 8044. The data link program is stored in a 2Kx8 EPROM (2732A), and a 2Kx8 static RAM (AM9128) is used as the external transmit and receive buffer. The RTS pin is connected to the CTS pin. For simplicity, the stations are assumed to be in the SDLC Normal Respond Mode after Hardware reset.

6.1.3 PRIMARY STATION SOFTWARE

The assembly code for the primary station software is listed in Appendix A. The primary software consists of the main routine, the SIU interrupt routine, and the receive interrupt routine. The receive interrupt routine is executed when a long frame is being received.

In the flow charts that follow, all actions taken by the SIU appear in squares, and actions taken by the on-chip CPU appear in spheres.

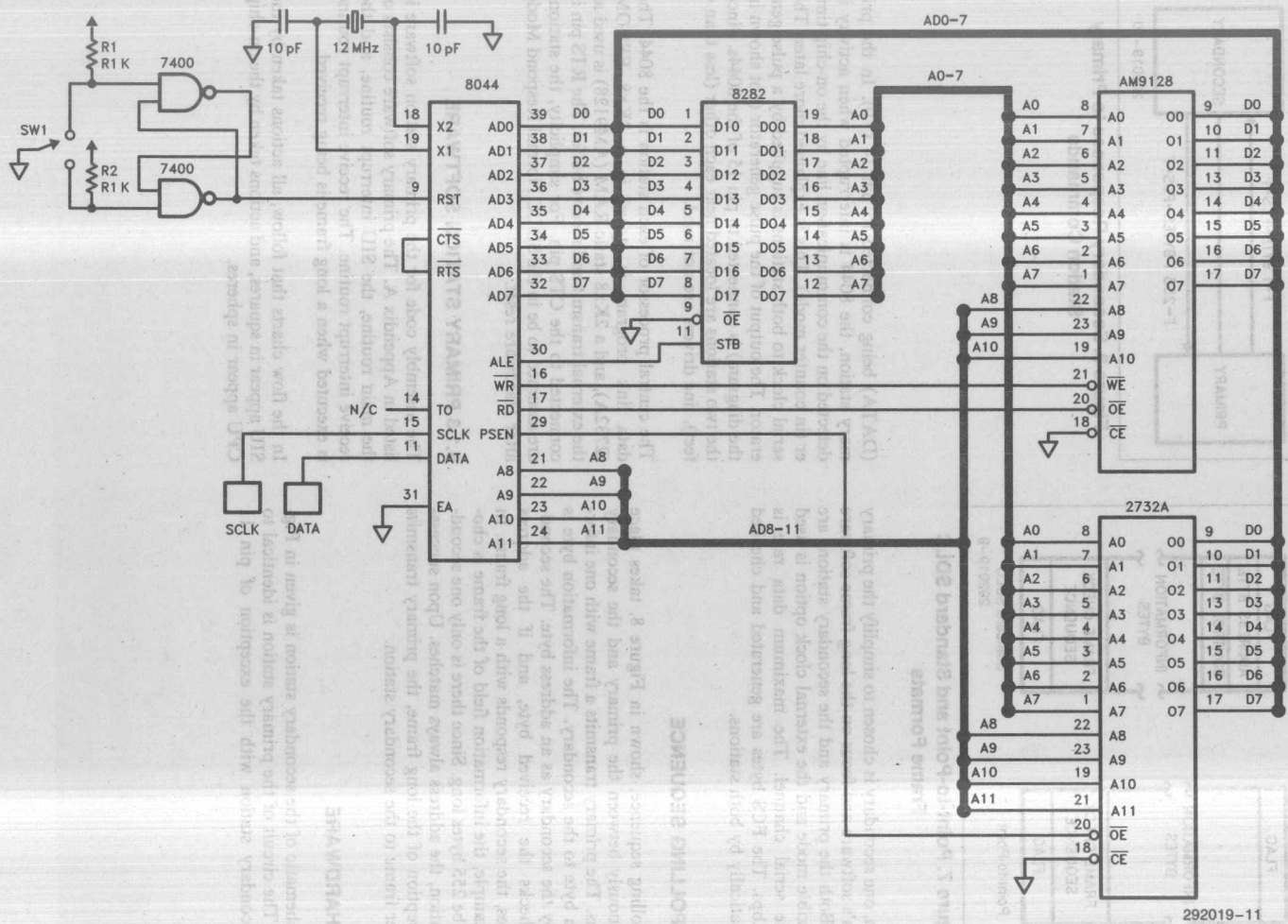


Figure 9. Secondary Station Hardware

Main Routine

First, the chip is initialized (see Figure 10). It is put in Flexible mode, externally clocked, and "Flag-Information Field-FCS-Flag" frame format. Pre-Frame Sync option (PFS = 1) and automatic Frame Check Sequence generation/detection (NFCS = 0) are selected. The on-chip transmit buffer starts at location 20H and the transmit buffer length is set to 1. This one byte buffer contains the address of the secondary station. There is no on-chip receive buffer since the long frame being received is moved to the external buffer. The RTS, TBF, and RBE bits are set simultaneously. Setting the RTS and TBF bits causes the SIU to transmit the contents of the transmit buffer.

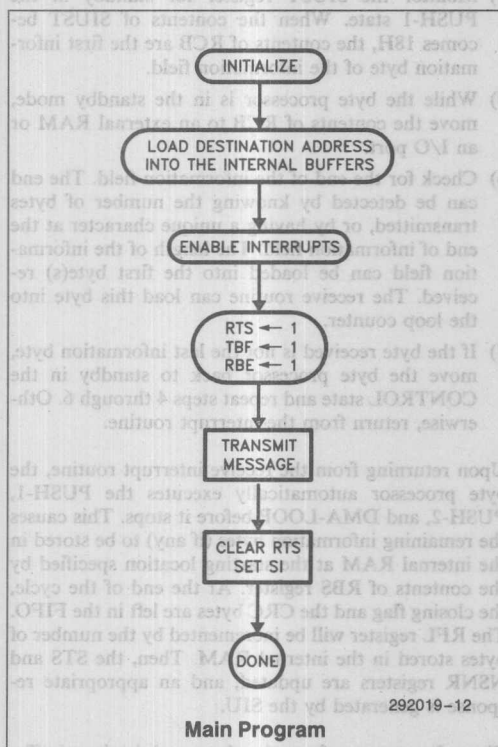


Figure 10. Primary Station Flow Charts

SIU Interrupt Routine

After transmission of the frame, the SIU interrupts the on-chip CPU (SI is set). In the SIU interrupt service routine, counter 0 is initialized and turned on (see Figure 11). The user program returns to perform other

tasks. After reception of the long frame, the SIU interrupt routine is executed again. This time, RTS, TBF, and RBE are set for another round of information exchange between the two stations.

SIU never interrupts while reception or transmission is taking place. The SIU registers are updated and the SI is set (serial interrupt) after the closing flag has been received or transmitted. An SIU interrupt never occurs if the receive interrupt routine or the transmit sub routine is being executed.

Setting the RBE bit of the STS register puts the RUPI in the receive mode. However, the jump to the receive interrupt routine occurs only when a frame appears on the serial port. Incoming frames can be detected using the Pre-Frame Sync. option and one of the CPU timers in counter mode. The counter external pin (T0) is connected to the data line (pin 11 is tied to pin 14). Setting the PFS (Pre-Frame Sync.) bit will guarantee 16 transitions before the opening flag of a frame.

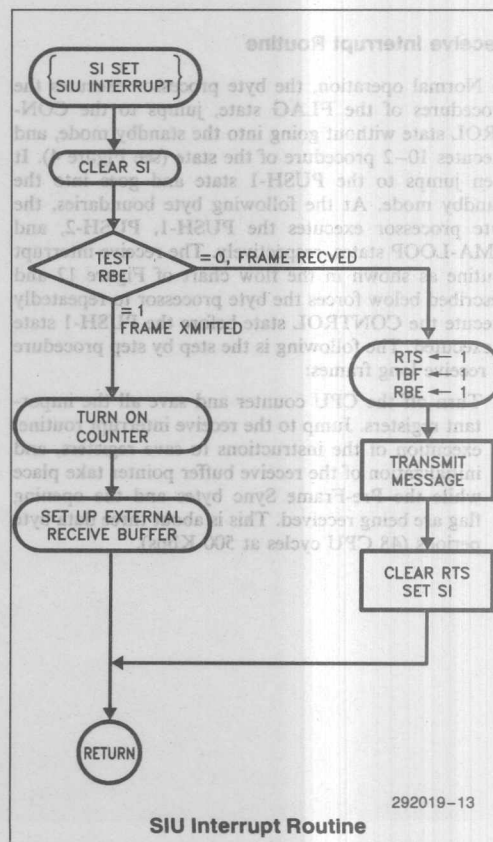


Figure 11. Primary Station Flow Charts

The counter registers are initialized such that the counter interrupt occurs before the opening flag of a frame. When the PFS transitions appear on the data line, the counter overflows and interrupts the CPU. The CPU program jumps to the timer interrupt service routine and executes the receive routine. In the receive routine, the received frame is processed, and the information bytes are moved to the external RAM. Note that the maximum count rate of the 8051 counter is $\frac{1}{24}$ of the oscillator frequency. At 12 MHz, the data rate is limited to 500 Kbps.

Another method to detect a frame on the data line and cause an interrupt is to use an external "Flag-Detect" circuit to interrupt the CPU. The "Flag Detect" circuit can be an 8-bit shift register plus some TTL chips. If this option is used, the RUPI must operate in externally clocked mode because the clock is needed to shift the incoming data into the shift register. With this option, the maximum data rate is not limited by the maximum count rate of the 8051 counter.

Receive Interrupt Routine

In Normal operation, the byte processor executes the procedures of the FLAG state, jumps to the CONTROL state without going into the standby mode, and executes 10-2 procedure of the state (see Figure 4). It then jumps to the PUSH-1 state and goes into the standby mode. At the following byte boundaries, the byte processor executes the PUSH-1, PUSH-2, and DMA-LOOP states, respectively. The receive interrupt routine as shown in the flow chart of Figure 12 and described below forces the byte processor to repeatedly execute the CONTROL state before the PUSH-1 state is executed. The following is the step by step procedure to receive long frames:

- 1) Turn off the CPU counter and save all the important registers. Jump to the receive interrupt routine, execution of the instructions to save registers, and initialization of the receive buffer pointer take place while the Pre-Frame Sync bytes and the opening flag are being received. This is about three data byte periods (48 CPU cycles at 500 Kbps).

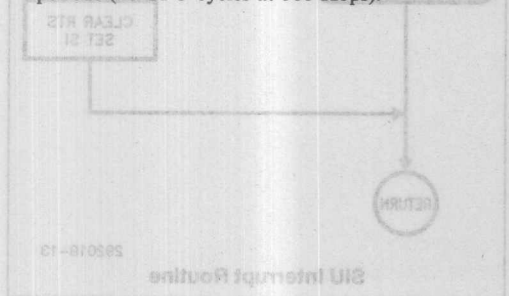


Figure 11. Primary Station Flow Charts

- 2) Monitor the SIUST register for standby in the PUSH-1 state (SIUST = 18H). When the SIUST contents are 18H, the byte processor is waiting for the first information byte. The bit processor has already recognized the flag and is processing the first information byte.
- 3) In the standby mode, move the byte processor into the CONTROL state by writing "EFH" (complement of 10H) into the SIUST register. When the next byte boundary occurs, the bit processor has processed and moved a byte of data into the SR register. The byte processor moves the contents of SR into the RCB register, jumps to the PUSH-1 state (SIUST = 18H), and waits.
- 4) Monitor the SIUST register for standby in the PUSH-1 state. When the contents of SIUST becomes 18H, the contents of RCB are the first information byte of the information field.
- 5) While the byte processor is in the standby mode, move the contents of RCB to an external RAM or an I/O port.
- 6) Check for the end of the information field. The end can be detected by knowing the number of bytes transmitted, or by having a unique character at the end of information field. The length of the information field can be loaded into the first byte(s) received. The receive routine can load this byte into the loop counter.
- 7) If the byte received is not the last information byte, move the byte processor back to standby in the CONTROL state and repeat steps 4 through 6. Otherwise, return from the interrupt routine.

Upon returning from the receive interrupt routine, the byte processor automatically executes the PUSH-1, PUSH-2, and DMA-LOOP before it stops. This causes the remaining information bytes (if any) to be stored in the internal RAM at the starting location specified by the contents of RBS register. At the end of the cycle, the closing flag and the CRC bytes are left in the FIFO. The RFL register will be incremented by the number of bytes stored in the internal RAM. Then, the STS and NSNR registers are updated, and an appropriate response is generated by the SIU.

The software to perform the above task is given in Table 1. In this example, the number of instruction cycles executed during standby is 12 cycles.

Table 1. Codes for Long Frame Reception

Receive Codes		Cycles
REC:	CLR	TRO
	MOV	A, #18H
WAIT1:	CJNE	A, SIUST, WAIT1
NEXTI:	MOV	SIUST, #0EFH
	MOV	A, #18H
WAIT2:	CJNE	A, SIUST, WAIT2
	MOV	A, RCB
	MOVX	@DPTR, A
	INC	DPTR
	DJNZ	R5, NEXTI
	RETI	
END		12 Cycles

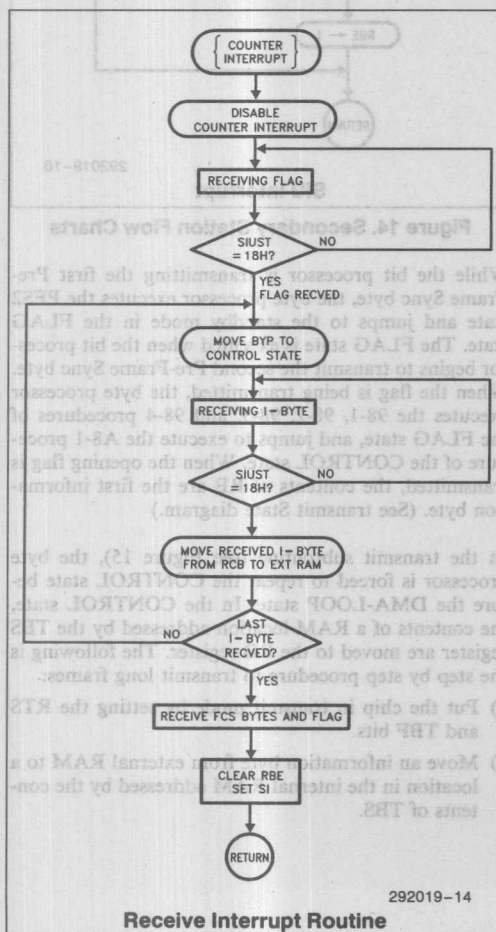


Figure 12. Primary Station Flow Charts

6.1.4 SECONDARY STATION SOFTWARE

The assembly code for the secondary station software is given in Appendix A. The secondary station contains the transmit subroutine which is called for transmission of long frames.

Main Routine

As shown in the secondary station flow chart (Figure 13), the external transmit buffer (external RAM) is loaded with the information data (FFH, FEH, FDH, ...) at starting location 200H. The internal transmit buffer (on chip RAM) starts at location 20H (TBS = 20H), and the transmit buffer length (TBL) is set to 1. The on-chip CPU, in the transmit subroutine, moves the information bytes from the external RAM to this one byte buffer for transmission. The receive buffer starts at location 10H and the receiver buffer length is 1. This buffer is used to buffer the frame transmitted by the primary. The received byte is used as an address byte.

The Secondary is configured like the Primary station. It is put in Flexible mode, externally clocked, Point-to-point frame format. The PFS bit is set to transmit two bytes before the first flag of a frame. The RBE bit is set to put the chip in receive mode. Upon reception of a valid frame, the SIU loads the received information byte into the on-chip receive buffer and interrupts the CPU.

SIU Interrupt Routine

In the serial interrupt routine, the RBE bit is checked (see Figure 14). Since RBE is clear, a frame has been received. The received Information byte is compared with the contents of the Station Address (STAD) register.

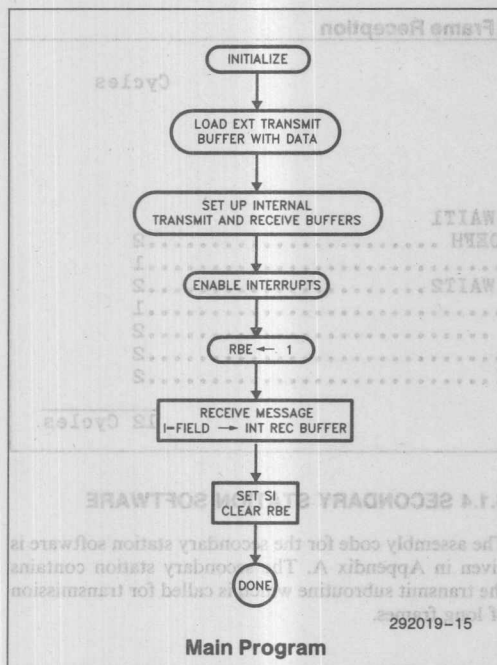


Figure 13. Secondary Station Flow Charts

If they match, the secondary will call the transmit subroutine to transmit the long frame. Upon returning from the transmit subroutine, the RBE bit is set, and program returns from the SIU interrupt. After transmission of the closing flag, SIU interrupt occurs again. In the interrupt routine, the RBE is checked. Since the RBE is set, the program returns from the SIU interrupt routine and waits until another long frame is received.

If the secondary were in Auto mode, the chip must be ready to execute the transmit routine upon reception of a poll-frame; otherwise, the chip automatically transmits the contents of the internal transmit buffer if the TBF bit is set, or transmits a supervisory command (RR or RNR) if TBF is clear.

Transmit Subroutine

In Normal operation the byte processor executes the START-TRANSMIT state and jumps to the PFS1 state. While the bit processor is transmitting some unwanted bits, the byte processor executes the PFS1 state and jumps to the standby mode in the PFS2 state.

In the serial interrupt routine, the RBE bit is checked (see Figure 14). Since RBE is clear, a frame has been received. The received information byte is compared with the contents of the Station Address (STAD) register.

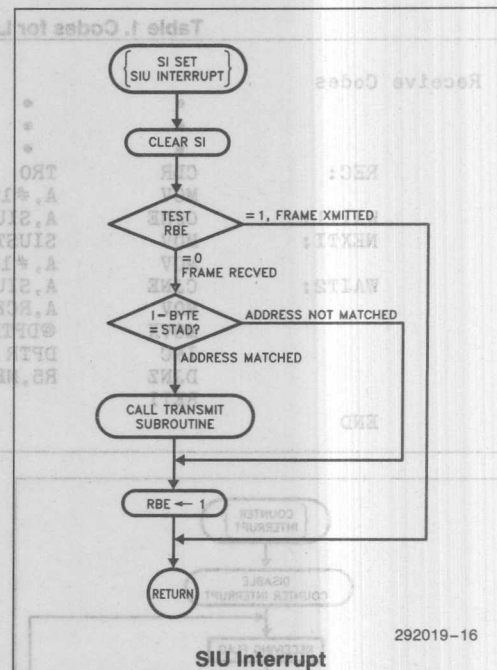


Figure 14. Secondary Station Flow Charts

While the bit processor is transmitting the first Pre-Frame Sync byte, the byte processor executes the PFS2 state and jumps to the standby mode in the FLAG state. The FLAG state is executed when the bit processor begins to transmit the second Pre-Frame Sync byte. When the flag is being transmitted, the byte processor executes the 98-1, 98-2, 98-3, and 98-4 procedures of the FLAG state, and jumps to execute the A8-1 procedure of the CONTROL state. When the opening flag is transmitted, the contents of RB are the first information byte. (See transmit State diagram.)

In the transmit subroutine (see Figure 15), the byte processor is forced to repeat the CONTROL state before the DMA-LOOP state. In the CONTROL state, the contents of a RAM location addressed by the TBS register are moved to the RB register. The following is the step by step procedure to transmit long frames:

- 1) Put the chip in transmit mode by setting the RTS and TBF bits.
- 2) Move an information byte from external RAM to a location in the internal RAM addressed by the contents of TBS.

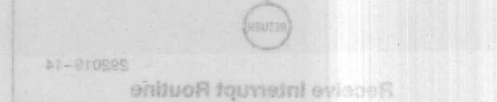


Figure 15. Primary Station Flow Charts

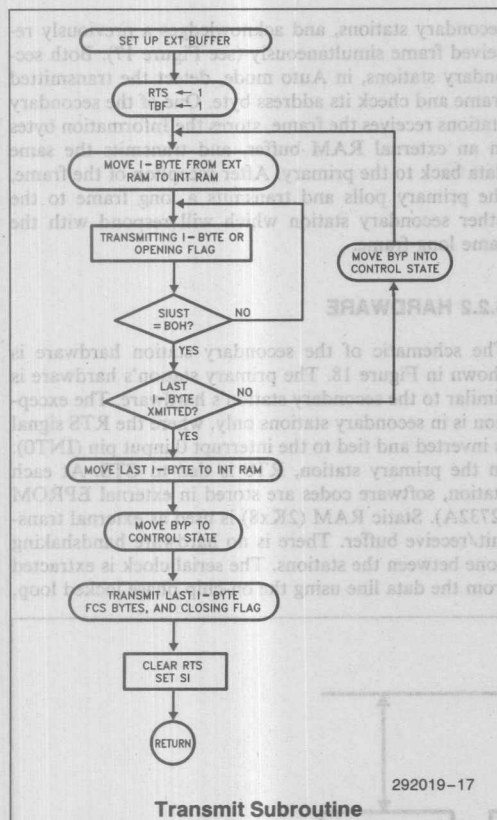


Figure 15. Secondary Station Flow Charts

Table 2. Codes for Long Frame Transmission

Transmit Codes		Cycles
TRAN:	MOV DPTR, #200H	
	MOV R5, #0FFH	
	SETB TBF	
LOOP:	SETB RTS	
	MOVX A, @DPTR	
	MOV @R1, A	
	MOV A, #0BOH	
WAIT1:	CJNE A, SIUST, WAIT1	2
	INC DPTR	2
	MOVX A, @DPTR	2
	MOV @R1, A	1
	DJNZ R5, NEXTI	2
	MOV SIUST, #57H	
	RET	
NEXTI:	MOV SIUST, #57H	2
	MOV A, #0BOH	1
	JMP WAIT1	1
END		
		13 Cycles

- 3) Monitor the SIUST register for the standby mode in the DMA-LOOP state (SIUST = B0H). When SIUST is B0H, the opening flag has been transmitted, and the first information byte is being transmitted by the bit processor.
- 4) If there are more information bytes, move the byte processor back to the CONTROL state, and repeat steps 2 through 4. Otherwise, continue.
- 5) Move byte processor to the Standby mode in the CONTROL state (SIUST = A8H) and return from the subroutine.

The byte processor automatically executes the remaining states to send the FCS bytes and the closing flag. After the completion of transmission, SIU updates the STS and NSNR registers and interrupts the CPU.

If the contents of the TBL register were more than 1, the SIU transmits (TBL) - 1 additional bytes from the internal RAM at starting address (TBS) + 1 because it executes the DMA-LOOP state (TBL) - 1 additional times. The byte processor should not be programmed to skip the DMA-LOOP state, because the transmission of FCS bytes is enabled in this state.

The maximum baud rate that can be used with these codes is calculated by adding the number of instruction cycles executed, during the standby mode, between each byte boundaries (see Table 2).

Using Equation 1, the maximum data rate, based on the transmit software, is 509 Kbps; However, the maximum count rate of the counter limits the data rate to 500 Kbps.

6.2 Multidrop Application

Performance of long frame in addition to the features of the 8044 are described using a simple multidrop communication system in which three RUPs, one as a master and the other two as secondary stations, transmit and receive long frames alternately (see Figure 16). All stations perform automatic zero bit insertion/deletion, NRZI decoding/encoding, Frame Check Sequence (FCS) generation/detection, and on-chip clock recovery at a data rate of 375 Kbps.

The primary and the secondary station's software code is given in Appendix B. These programs, for simplicity, assume only reception of information and supervisory frames. It is also assumed that the frames are received and transmitted in order. All stations use very similar transmit and receive routines. This code is written for standard SDLC frames (see Figure 7).

6.2.1 POLLING SEQUENCE

The primary station, in Flexible mode, transmits a long frame (for this example, 255 I-bytes), polls one of the

secondary stations, and acknowledges a previously received frame simultaneously (see Figure 17). Both secondary stations, in Auto mode, detect the transmitted frame and check its address byte. One of the secondary stations receives the frame, stores the Information bytes in an external RAM buffer, and transmits the same data back to the primary. After reception of the frame, the primary polls and transmits a long frame to the other secondary station which will respond with the same long frame.

6.2.2 HARDWARE

The schematic of the secondary station hardware is shown in Figure 18. The primary station's hardware is similar to the secondary station's hardware. The exception is in secondary stations only, where the RTS signal is inverted and tied to the interrupt 0 input pin (INT0). In the primary station, RTS is tied to CTS. At each station, software codes are stored in external EPROM (2732A). Static RAM (2Kx8) is used as external transmit/receive buffer. There is no hardware handshaking done between the stations. The serial clock is extracted from the data line using the on-chip phase locked loop.

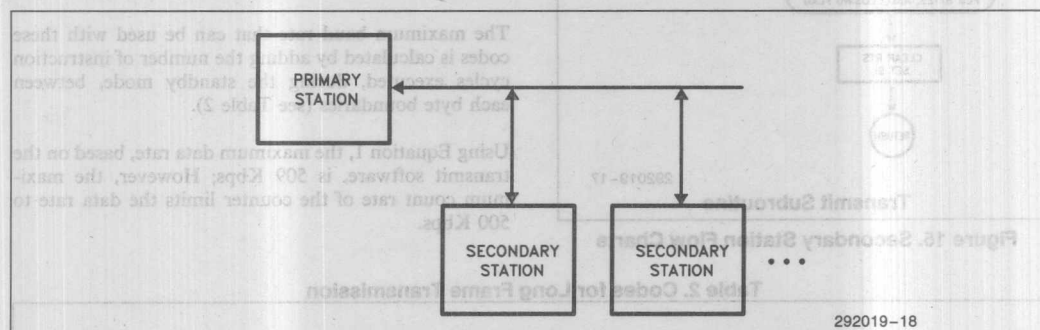


Figure 16. SDLC Multidrop Application Example

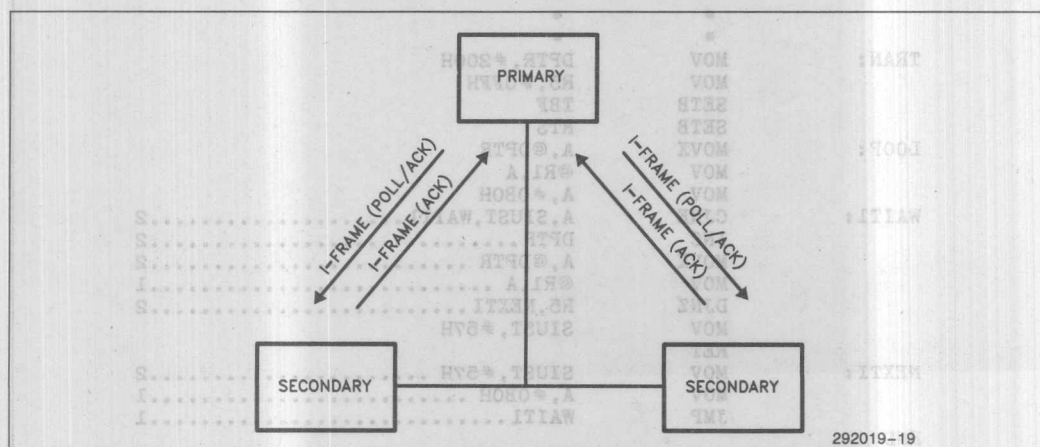


Figure 17. Polling Sequence Between the Primary and Secondary Stations



Figure 18. Secondary Station Hardware

6.2.3 PRIMARY SOFTWARE

Main Routine

During initialization (see Figure 19), the 8044 is set to Flexible mode, internally clocked at 375 Kbps, and configured to handle standard SDLC frames. The on-chip receive and transmit buffer starting addresses and lengths are selected. The external transmit buffer is chosen from physical location 200H to location 2FFH (255 bytes). The external transmit buffer (external RAM) is loaded with data (FFH, FEH, FDH, FCH, ... 00H). Timer 0 is put in counter mode and set to priority 1. The counter register (TL0) is loaded such that interrupt occurs after 8 transitions on the data line. The Pre-Frame Sync option (setting bit 2 of the SMD register) is selected to guarantee at least 16 transitions before the opening flag of a frame.

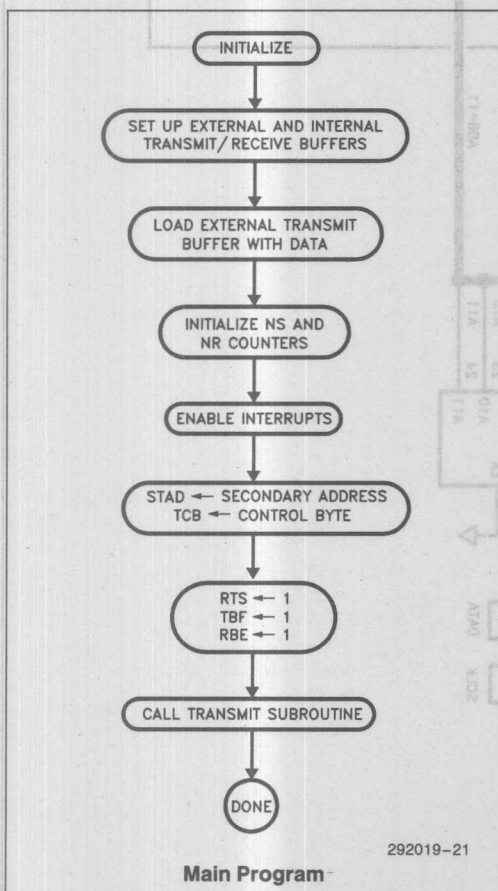


Figure 19. Primary Station Flow Charts

The station address register (STAD) is loaded with address of one of the secondary stations. The RTS, TBF, and RBE bits of the STS register are simultaneously set and a call to the transmit routine follows. The transmit routine transmits the contents of the external transmit buffer. At the end of transmission, RTS and TBF are cleared by the SIU, and SIU interrupt occurs. In Flexible mode, SIU interrupt occurs after every transmission or reception of a frame.

SIU Interrupt Routine

In the SIU interrupt service routine (see Figure 20), SI is cleared and the RBE bit is checked. If RBE is set, a long frame has been transmitted. The first time through the SIU interrupt service routine, the RBE test indicates a long frame has been transmitted to one of the secondary stations. Therefore, the Counter is initialized

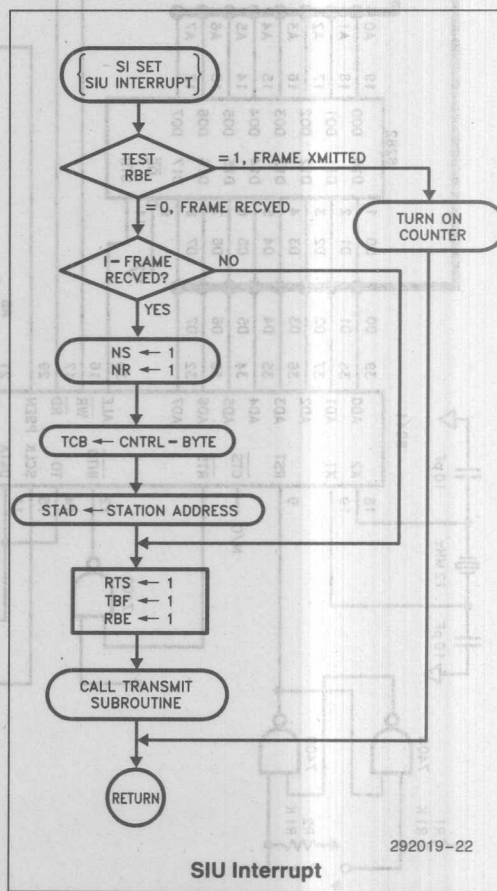


Figure 20. Primary Station Flow Charts

and turned on. The program returns from the interrupt routine before a frame appears on the communication channel.

When a frame appears on the communication line, counter interrupt occurs and the receive routine is executed to move the incoming bytes into the external RAM. After reception of the frame and return from the receive routine, SIU interrupt occurs again.

In the SIU interrupt routine, RBE is checked. Since the RBE bit is clear, a frame has been received. Therefore, the appropriate NS and NR counters are incremented and loaded into the TCB register (two pairs of internal RAM bytes keep track of NS and NR counts for the two secondary stations). Transmission of a frame to the next secondary station is enabled by setting the RTS and the TBF bits. The chip is also put in receive mode (RBE set), and a call to transmit routine is made. After transmission, SIU interrupt occurs again, and the process continues.

6.2.4 SECONDARY SOFTWARE

Main Routine

Both secondary stations have identical software (Appendix B). The only differences are the station addresses. Contents of the STAD register are 55H for one station and 44H for the other.

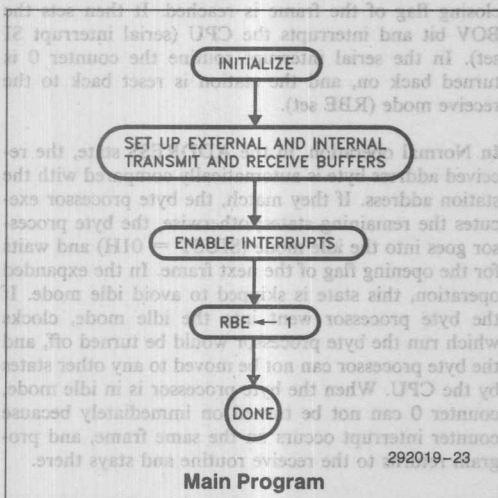


Figure 21. Secondary Station Flow Charts

During initialization, the chip is set to Auto mode, standard SDLC frame, and internally clocked at 375 Kbps (see Figure 21). Internal buffer registers: RBS, RBL, TBS, and TBL are initialized. The RBE bit is set and the counter 0 is turned on.

The secondary is configured to transmit an Information frame every time it is polled. The RTS pin is inverted and tied to INT1 pin. External interrupt 1 is enabled and set to interrupt on low to high transition of the RTS signal. This will cause an interrupt (EX1 set) after a frame is transmitted. In the interrupt routine the CTS pin is cleared to prevent any automatic response from the secondary. If the CTS pin were not disabled, the secondary station would respond with a supervisory frame (RNR) since the TBF is set to zero by the SIU due to the acknowledge. In the SIU interrupt routine, the CTS pin is cleared after the TBF bit is set. If this option is not used, the primary should acknowledge the previously received frame and poll for the next frame in two separate transmissions.

SIU Interrupt Routine

When a frame is received, counter 0 interrupt occurs and the receive routine is executed (see Figure 22). If the incoming frame is addressed to the station, the information bytes are stored in external RAM; otherwise, the program returns from the receive routine to perform other tasks. At the end of the frame, SIU interrupt occurs. In Auto mode, SIU interrupt occurs whenever an Information frame or a supervisory frame is received. Transmission will not cause an interrupt. In the SIU interrupt service routine, the AM bit of the STS is checked.

If AM bit is set, the interrupt is due to a frame whose address did not match with the address of the station. In this case, NFCS, AM, and the BOV bits are cleared, the RBE bit is set, the counter 0 is initialized and turned on, and program returns from the interrupt routine.

If AM bit is not set, a valid frame has been received and stored in the external RAM. TBF bit is set, CTS pin is activated, counter 0 is disabled and a call to transmit routine is made which transmits the contents of external transmit buffer. This frame also acknowledges the reception of the previously received frame (NS and NR are automatically incremented). Upon return from the transmit routine RBE is set and counter 0 is turned on, thereby putting the chip in the receive mode for another round of data exchange with the primary.

Note that, if the second station is in receive mode, and the counter is enabled and turned on, the CPU will be interrupted each time a frame is on the communication channel. If the frame is not addressed to the secondary station, the chip enters the receive routine, executes only a few lines of code (address comparison) and returns to perform other tasks. This interrupt will not occupy the CPU for more than two data byte periods (43 microseconds at 375 Kbps). At the end of the frame, the BOV bit is set by the SIU, and the SIU interrupt occurs. In the SIU interrupt service routine,

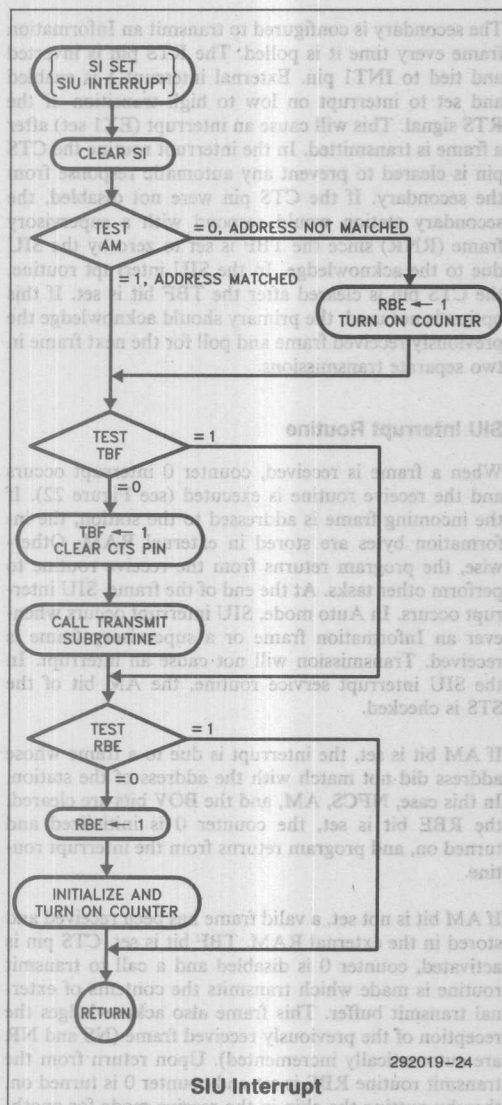


Figure 22. Secondary Station Flow Charts

the RBE bit is set and the counter is turned on which put the chip back in the receive mode.

6.2.5 RECEIVE INTERRUPT ROUTINE

Assembly code for the receive interrupt routine can be found in both primary and secondary software (Appendix B). The receive interrupt routine of the primary station is very similar to that of the primary station in example 1. In the following two sections the receive and transmit routine of the secondary stations are discussed.

In the receive interrupt service routine (see Figure 23), counter 0 is turned off, important registers are saved, receive buffer starting address and receive buffer length of the external RAM are set (do not confuse the external RAM settings with that of the internal RAM buffer.)

After reception of an opening flag, the byte processor jumps to the ADDRESS state and waits until the bit processor processes and moves the receiving address byte to SR. Then, the byte processor is triggered to execute the state. In the secondary stations, the CPU monitors the SIUST register for the ADDRESS state (SIUST = 08H). When the ADDRESS state is reached, the byte processor is moved to the next state (CONTROL state), and the ADDRESS state is skipped. Therefore, when the address byte is moved to SR, the byte processor executes the CONTROL state rather than the ADDRESS state and then jumps to the PUSH-1 state. The execution of the CONTROL state causes the contents of SR (the received address byte) to be loaded into the RCB register.

The CPU checks the contents of RCB with the contents of the STAD (Station Address) register. If they match, the receive routine continues to store the received Information bytes in the external RAM buffer; Otherwise, the byte processor is moved to the very last state (BOV-LOOP), and the program returns from the routine to perform other tasks. The byte processor executes the BOV-LOOP state in each byte boundary until the closing flag of the frame is reached. It then sets the BOV bit and interrupts the CPU (serial interrupt SI set). In the serial interrupt routine the counter 0 is turned back on, and the station is reset back to the receive mode (RBE set).

In Normal operation, in the ADDRESS state, the received address byte is automatically compared with the station address. If they match, the byte processor executes the remaining states; otherwise, the byte processor goes into the idle mode (SIUST = 01H) and waits for the opening flag of the next frame. In the expanded operation, this state is skipped to avoid idle mode. If the byte processor went into the idle mode, clocks which run the byte processor would be turned off, and the byte processor can not be moved to any other states by the CPU. When the byte processor is in idle mode, counter 0 can not be turned on immediately because counter interrupt occurs on the same frame, and program returns to the receive routine and stays there.

If the address byte matches the station address, the byte processor is moved to the CONTROL state again. This time, after execution of the CONTROL state the contents of RCB are the received control byte.

CPU investigates the type of received frame by checking the received control byte. If the receiving frame is not an information frame (i.e. Supervisory frame), execution of receive routine will be terminated to free the

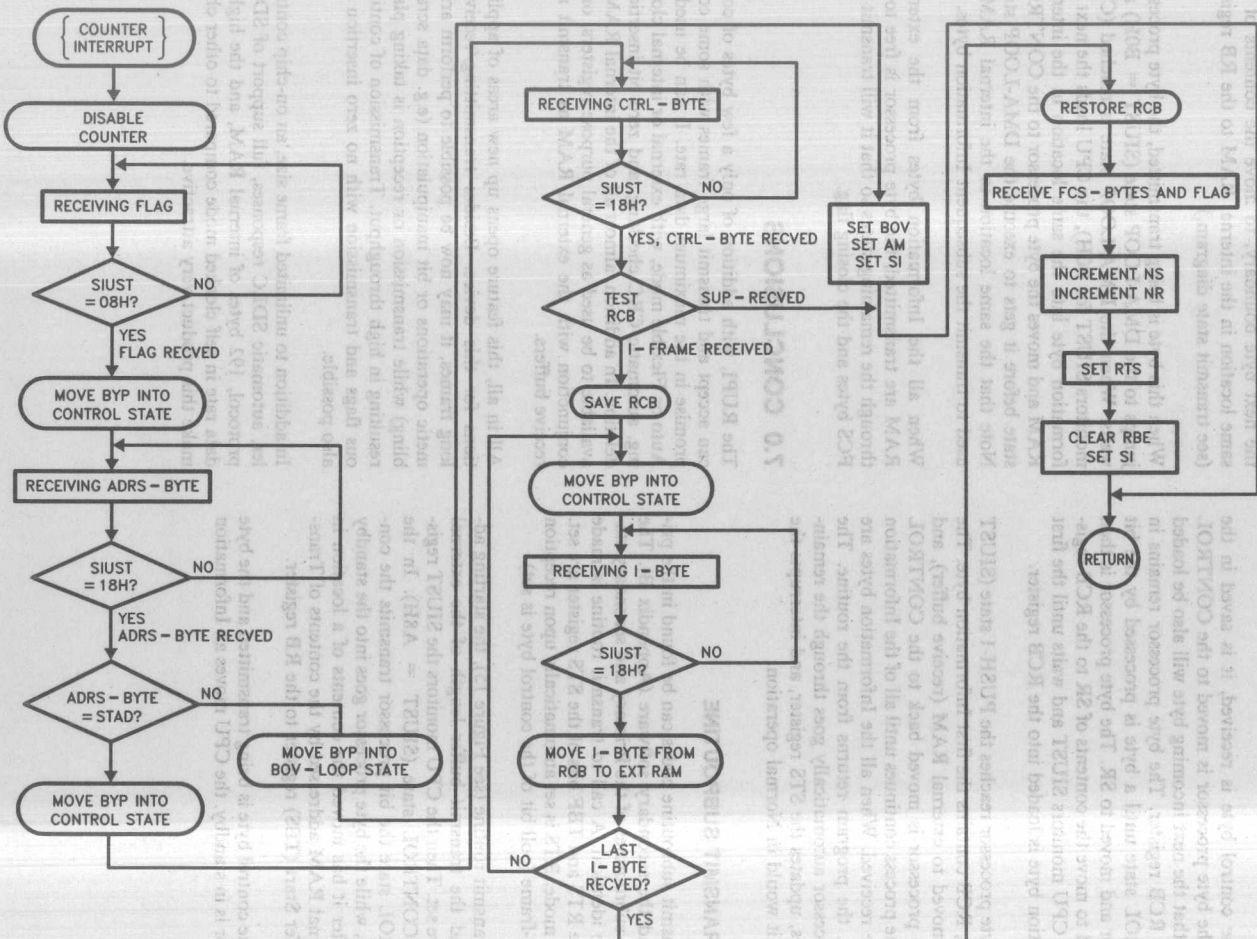


Figure 23. Receive Flow Chart (secondary station)

APPENDIX A

LISTING OF SOFTWARE MODULES

FOR APPLICATION EXAMPLE 1

```
$DEBUG NOMOD51
$INCLUDE (REG44.PDF)
```

```
; ASSEMBLY CODE FOR PRIMARY STATION (POINT TO POINT)
; FLEXIBLE MODE; FCS OPTION
```

```
ORG 00H ; LOCATIONS 00 THRU 26H ARE USED
SJMP INIT ; BY INTERRUPT SERVICE ROUTINES.
ORG 0BH ; VECTOR ADDRESS FOR TIMERO INT.
JMP REC ;
ORG 23H ; VECTOR ADDRESS FOR SIU INT.
SJMP SIINT
```

```
;***** INITIALIZATION *****
```

```
INIT: ORG 26H
MOV SMD,#00000110B ; EXT CLOCK; PFS=NB=1
MOV TBS,#20H ; INT TRANSMIT BUFFER START
MOV TBL,#01H ; INT TRANSMIT BUFFER LENGTH
MOV 20H,#55H ; STATION ADDRESS
MOV TMOD,#00000111B ; COUNTER FUNCTION; MODE 3
MOV IE,#10010010B ; EA=1; SI=1; ET0=1
MOV STS,#11100000B ; TRANSMIT A FRAME
DOT: SJMP DOT ; WAIT FOR AN INTERRUPT
```

```
; SIU TRANSMITS THE PFS BYTES, THE OPENNING FLAG, THE CONTENTS
; OF LOCATION 20H, THE CALCULATED FCS-BYTES, AND THE CLOSING
; FLAG. AT THE END OF TRANSMISSION, SIU INTERRUPT OCCURS.
```

```
;***** SERIAL CHANNEL INTERRUPT ROUTINE *****
```

```
SIINT: CLR SI ; TRANSMITTED A FRAME ?
JNB RBE,RECVD ; YES, INITIALIZE COUNTER REGISTER
MOV TLO,#0F8H ; EXT RAM RECEIVE BUFFER START
MOV DPTR,#200H ; EXT RAM RECEIVE BUFFER LENGTH
SETB TRO ; TURN ON COUNTER 0
RETI ; RETURN
```

```
; WHEN A FRAME APPEARS ON THE SERIAL CHANNEL, COUNTER (RECEIVE)
; INTERRUPT OCCURS. AFTER SERVICING THE INTERRUPT ROUTINE, SIU
; INTERRUPT OCCURS.
```

```
RECVD: MOV STS,#11100000B ; TRANSMIT A FRAME
RETI ; RETURN
```

```
;***** RECEIVE INTERRUPT ROUTINE *****
```

```
REC: CLR TRO ; DISABLE THE COUNTER 0 INTERRUPT
MOV A,#18H ; PUSH-1 STATE
WAIT1: CJNE A,S1UST,WAIT1 ; MOVE BYP TO CONTROL STATE
NEXT1: MOV A,#18H ; PUSH-1 STATE
WAIT2: CJNE A,S1UST,WAIT2 ;
MOV A,RCB ; MOVE RECEIVED BYTE INTO ACC.
MOVX @DPTR,A ; MOVE DATA TO EXT. RAM
INC DPTR ; INCREMENT POINTER TO EXT RAM
DJNZ R5,NEXT1 ; LAST BYTE RECEIVED?
RETI ; YES, RETURN
```

```
END
```

292019-29

```
$DEBUG NOMOD51
$INCLUDE (REG44.PDF)

; ASSEMBLY CODE FOR SECONDARY STATION (POINT TO POINT)
; FLEXIBLE MODE; FCS OPTION

ORG 00H
SJMP INIT
ORG 23H
SJMP SIINT

;***** LOAD TRANSMIT BUFFER WITH DATA *****
```

```
ORG 26H
INIT: MOV DPTR,#200H ; EXT RAM XMIT BUFFER START
      MOV R3,#0FFH ; EXT RAM XMIT BUFFER LENGHT
LDRAM: MOV A,R3
      MOVX @DPTR,A ; LOAD EXT BUFFER WITH FFH,FEH,...
      INC DPTR ; INCREMENT POINTER
      DJNZ R3,LDRAM
```

```
;*****INITIALIZATION *****
```

```
MOV SMD,#00000110B ; EXT CLOCK; PFS=NB=1
MOV R1,#10H
MOV TBS,R1 ; INT RAM XMIT BUFFER START
MOV TBL,#01H ; INT RAM XMIT BUFFER LENGTH
MOV RBS,#20H ; INT RAM RECEIVE BUFFER START
MOV RBL,#01H ; INT RAM RECEIVE BUFFER LENGTH
MOV STAD,#55H ; STAD ADDRESS=55H
MOV TCON,#00H ; RESET TCON REGISTER
MOV IE,#10010000B ; ENABLE SI INT. ;EA=1
MOV IP,#0FFH ; ALL INTERRUPTS: PRIORITY 1
MOV SPS,#01000000B ; RBE=1, RECEIVE A FRAME.
DOT: SJMP DOT ; WAIT FOR AN INTERRUPT
```

```
; SIU INTERRUPT OCCURS AT THE END OF A RECEIVED FRAME OR
```

```
; A TRANSMITTED FRAME.
```

```
;***** SERIAL CHANNEL INTERRUPT ROUTINE *****
```

```
SIINT: CLR SI
      JB RBE,RETRN ; RECEIVED A FRAME?
      MOV A,STAD ; YES
      CJNE A,20H,NMACH ; STATION ADDRESS MATCHED?
      ACALL TRAN ; YES, CALL TRANSMIT SUBROUTINE
```

```
; TRANSMIT SUBROUTINE IS CALLED TO TRANSMIT A LONG FRAME.
```

```
; AFTER TRANSMISSION, SI IS SET. SIU INTERRUPT IS SERVICED
```

```
; AFTER THE CURRENT ROUTINE (SIINT) IS COMPLETED.
```

```
NMACH: SETB RBE ; RBE=1, RECEIVE A FRAME
      RETN ; RETURN
```

```
;***** TRANSMIT SUBROUTINE *****
```

```
TRAN: MOV DPTR,#200H ; EXT RAM RECEIVE BUFFER START
      MOV R5,#0FFH ; EXT RAM RECEIVE BUFFER LENGTH
      SETB TBF ; SET TRANSMIT BUFFER FULL
      SETB RTS ; ENABLE XMISSION OF AN I-FRAME
      MOVX A,@DPTR ; MOVE THE 1ST I-BYTE INTO ACC.
      MOV @R1,A ; THEN, MOVE TO INT. RAM @ (TBS)
      MOV A,#0B0H ; DMA-LOOP STATE
      WAIT1: CJNE A,SIUST,WAIT1 ; WAIT FOR XMISSION OF AN I-FRAME
      INC DPTR ; INCREMENT POINTER TO EXT. RAM
      DJNZ R5,NEXTI ; ALL BYTES XMITTED?
      MOVX A,@DPTR ; YES, EXCEPT THE LAST BYTE.
      MOV @R1,A ; MOVE DATA INTO INT. RAM @ (TBS)
      MOV SIUST,#57H ; MOVE BYP TO CONTROL STATE
      RET ; THE SIU TRANSMITS THE FCS-BYTES
      ; AND THE CLOSING FLAG.
      ; RETURN
NEXTI: MOV SIUST,#57H ; MOVE BYP TO CONTROL STATE (ASH).
      JMP LOOP ; TRANSMIT THE NEXT BYTE
END
```

292019-31

APPENDIX B

LISTING OF SOFTWARE MODULES FOR APPLICATION EXAMPLE 2

```
$DEBUG NOMODS1
$INCLUDE (REG44.PDF)
```

```
; ASSEMBLY CODE FOR PRIMARY STATION (MULTIPOINT)
; FLEXIBLE MODE; FCS OPTION
```

```
ORG 00H ; LOCATIONS 00 THRU 26H ARE USED
SJMP INIT ; BY INTERRUPT SERVICE ROUTINES.
ORG 0BH ; VECTOR ADDRESS FOR TIMERO INT.
JMP REC ;
ORG 23H ; VECTOR ADDRESS FOR SIU INT.
SJMP SIINT
```

```
;***** LOAD TRANSMIT BUFFER WITH DATA *****
```

```
INIT: MOV 26H ; EXT RAM XMIT BUFFER START
MOV DPTR,#200H ; EXT RAM XMIT BUFFER LENGHT
LDRAM: MOV A,R3
MOVX @DPTR,A ; LOAD BUFFER WITH FFH,FEH,...00
INC DPTR ; INCREMENT POINTER
DJNZ R3,LDRAM
```

```
;***** INITIALIZATION *****
```

```
LOOP: MOV RO,#0BFH ; PUT ZEROS INTO INT. RAM
MOV A,#00H ; FROM BFH TO 40H.
MOV @RO,A ; MOVE 0 INTO RAM ADDRESSD BY RO
DEC RO
CJNE RO,#40H,LOOP
```

```
MOV 30H,#00H ; NS COUNTER FOR STAD=55
MOV 31H,#00H ; NR COUNTER FOR STAD=55
MOV 32H,#0FFH ; NS COUNTER FOR STAD=44
MOV 33H,#0FFH ; NR COUNTER FOR STAD=44
MOV 34H,#01H ; POINTER TO SECONDARY STATIONS
MOV SMD,#11010100B ; INT. CLKED @ 375K; NRZ1=1; PFS=1
MOV RBS,#10H ; INT. RAM RECEIVE BUFFER START=10H
MOV RBL,#00H ; INT. RAM RECEIVE BUFFER LENGTH=0
MOV R1,#20H ; INT. RAM XMIT BUFFER START=20H
MOV TBS,R1
MOV TBL,#01H ; INT. RAM XMIT BUFFER LENGTH=1
MOV NSNR,#00H ; NS-NR=0
MOV TMOD,#00000111B ; COUNTER FUNCTION, MODE 3
MOV TCON,#00H
MOV IE,#10010010B ; EA=1; SI=1; ETO=1
MOV IP,#00000010B ; TIMER 0 INT. PRIORITY 1
MOV TCB,#00010000B ; I-FRAME W/POLL
MOV STAD,#55H ; ADDRESS BYTE=55H
MOV STS,#11100000B ; RBE-TBF=RTS=1
```

```
; TRANSMIT A LONG FRAME WITH POLL BIT SET, WAIT FOR A
; RESPONSE.
```

```
DOT: ACALL TRAN ; CALL TRANSMIT ROUTINE
SJMP DOT ; WAIT FOR AN INTERRUPT
```

292019-32

292019-33

;***** SERIAL INTERRUPT ROUTINE *****

```

SIINT: CLR SI ; CLEAR SI
      JB RBE,RETURN ; RECEIVED A FRAME ?
      MOV A,RCB ; YES, LOAD ACC WITH REC CNTRL BYTE
      JB ACC.0,GETI ; IS IT AN I-FRAME ?
      MOV A,#01H ; YES
      CJNE A,34H,SKIP ;
      MOV A,30H ; MOVE NS INTO ACC.
      INC A ; INCREMENT NS
      ANL A,#00000111B ; MASK OUT THE LEAST 3 SIG. BITS
      MOV 30H,A ; SAVE NS
      MOV A,31H ; MOVE NR INTO ACC.
      INC A ; INCREMENT NR
      ANL A,#00000111B ; MASK OUT THE LEAST 3 SIG. BITS
      MOV 31H,A ; SAVE NR
      RL A ; SHIFT 4 BITS TO LEFT
      RL A
      RL A
      ORL A,30H ; MOVE NS COUNT TO ACC.
      RL A ; SHIFT 1 BIT TO LEFT
      ORL A,#00010000B ; SET THE POLL BIT
      MOV TCB,A ; MOVE CONTROL BYTE INTO TCB REG.

```

TCB: NR2,NR1,NR0,1,NS2,NS1,NS0,0

```

      MOV STAD,#55H
      MOV 34H,#00H
      JMP GETI

SKIP: MOV A,32H ; MOVE NS INTO ACC.
      INC A ; INCREMENT NS
      ANL A,#00000111B ; MASK OUT THE LEAST 3 SIG. BITS
      MOV 32H,A ; SAVE NS
      INC A ; MOVE NR INTO ACC.
      ANL A,#00000111B ; MASK OUT THE LEAST 3 SIG. BITS
      MOV 33H,A ; SAVE NR
      RL A ; SHIFT 4 BITS TO LEFT
      RL A
      RL A
      ORL A,33H ; MOVE NS COUNT TO ACC.
      RL A ; SHIFT 1 BIT TO LEFT
      ORL A,#00010000B ; SET THE POLL BIT
      MOV TCB,A ; MOVE CONTROL BYTE INTO TCB

```

TCB: NR2,NR1,NR0,1,NS2,NS1,NS0,0

```

      MOV STAD,#44H
      MOV 34H,#01H
      MOV STS,#11100000B ; ENABLE TRANSMISSION
      ACALL TRAN ; CALL TRANSMIT ROUTINE
      RETI

```

```

RETURN: CLR EA ; DISABLE ALL INTERRUPTS
      MOV TLO,#0FBH ; INTERRUPT AFTER 8 COUNTS
      SETB TRO ; TURN ON COUNTER 0
      SETB EA
      RETI

```

;***** RECEIVE INTERRUPT ROUTINE *****

```

REC: CLR TRO ; TURN OFF COUNTER 0
      MOV DPTR,#400H ; EXT. RAM RECEIVE BUFFER START
      MOV R5,#0FFH ; EXT. RAM RECEIVE BUFFER LENGTH
      MOV A,#18H ; PUSH-1 STATE
      WAIT1: CJNE A,SIUST,WAIT1 ; WAIT FOR THE CONTROL BYTE
      PUSH RCB ; SAVE RECEIVE CONTROL BYTE
      NEXTI: MOV SIUST,#0EFH ; PUSH "BYP" INTO CONTROL STATE(10H).
      MOV A,#18H ; PUSH-1 STATE
      WAIT2: CJNE A,SIUST,WAIT2 ; WAIT FOR AN I-BYTE
      MOV A,RCB ; MOVE RECEIVED I-BYTE INTO ACC.
      MOVX @DPTR,A ; MOVE DATA TO EXT. RAM
      INC DPTR ; INCREMENT PTR TO EXTERNAL RAM
      DJNZ R5,NEXTI ; IS IT THE LAST I-BYTE?
      POP RCB ; YES, RESTORE THE CONTENTS OF RCB
      RETI ; RETURN

```

;***** TRANSMIT SUBROUTINE *****

```

TRAN: MOV DPTR,#200H ; EXT. RAM TRANSMIT BUFFER START
      MOV R5,#0FFH ; EXT. RAM TRANSMIT BUFFER LENGTH
      MOV A,#0A8H ; CONTROL STATE
      WAIT: CJNE A,SIUST,WAIT ; WAIT FOR CTRL BYTE XMISSION
      MOVX @DPTR,A ; MOVE DATA FROM EXT. RAM TO ACC.
      INC DPTR ; MOVE DATA INTO INT. RAM @ (TBS)
      DJNZ R5,NXTI ; INCREMENT POINTER
      MOV SIUST,#57H ; IS IT THE LAST I-BYTE ?
      RET ; NO. XMIT THE LAST I-BYTE
      NXTI: MOV SIUST,#57H ; KEEP "BYP" IN CONTROL STATE(A8H).
      MOV A,#0B0H ; DMA-LOOP STATE
      JMP WAIT ; TRANSMIT THE NEXT BYTE

```

END

292019-36


```
$DEBUG NOMOD51
$INCLUDE (REG44.PDF)
```

```
; ASSEMBLY CODE FOR SECONDARY STATIONS: (MULTIPOINT)
; AUTO MODE; FCS OPTION
```

```
ORG 00H
SJMP INIT
ORG 0BH
JMP REC
ORG 13H
JMP XINT1
ORG 23H
JMP SIINT
```

```
;*****INITIALIZATION*****
```

```
INIT:  ORG 26H
        MOV SMD,#11010100B
        MOV STAD,#55H

        MOV RBS,#10H
        MOV RBL,#00H
        MOV RL,#20H
        MOV TBS,RL
        MOV TBL,#01H
        MOV NSNR,#00H
        MOV TCON,#00000100B
        MOV IE,#00010110B
        MOV IP,#00000010B
        MOV TMOD,#0000011B
        MOV STS,#01000010B
        MOV TLO,#0F8H
```

```
; STATE JUMPING ON "BYE" FROM
; INT. CLKED @ 375K; NRZI=1; PFS=1
; STATION ADDRESS; STAD=44H FOR THE
; OTHER STATION
; INT. RAM RECEIVE BUFFER START
; INT. RAM RECEIVE BUFFER LENGTH
; INT. RAM XMIT BUFFER START
; INT. RAM XMIT BUFFER LENGTH
; NS=NR=0
; EXT. INT.: EDGE TRIGGERED
; SI=1; ETO=1; EXO=1
; TIMER 0: PRIORITY 1
; COUNTER FUNCTION: MODE 3
; RECEIVE I-FRAME.
; SET COUNTER TO OVERFLOW
; AFTER 8 COUNTS
; TURN ON COUNTER
; ENABLE ALL INTERRUPTS
; WAIT FOR AN INTERRUPT.
```

```
SETB TRO
SETB EA
SUMP DOT
; CPU IS INTERRUPTED AT THE END OF RECEPTION (SI SET), AND AT*
; THE END OF LONG-FRAME TRANSMISSION (EXO SET). *
```

292019-37

```
;*****EXTERNAL INTERRUPT*****
```

```
XINT1: SETB P1.7
        RETI
; DISABLE CTS PIN
; RETURN.
```

```
;***** SERIAL INTERRUPT ROUTINE *****
```

```
SIINT:  CLR SI
        JB AM,HOP
        CLR EA
        MOV STS,#01000010B
        MOV TLO,#0F8H
        SETB TRO
        SETB EA
        RETI
; ADDRESS MATCHED?
; DISABLE ALL INTERRUPTS
; RBE=1; NB=1
; TURN ON COUNTER 0
; ENABLE ALL INTERRUPTS
; RETURN.
```

```
HOP:   JB TBF,GETI
        SETB TBF
        CLR P1.7
        ACALL TRAN
        GETI: JB RBE,RETURN
        CLR EA
        SETB RBE
        MOV TLO,#0F8H
        SETB TRO
        SETB EA
        RETURN: RETI
; A FRAME TRANSMITTED?
; ENABLE TRANSMISSION OF I-FRAME
; ENABLE CTS PIN
; CALL TRANSMIT ROUTINE
; A FRAME RECEIVED?
; DISABLE ALL INTERRUPTS
; PUT RUP1 IN RECEIVE MODE
; TURN ON COUNTER 0
; ENABLE ALL INTERRUPTS
; RETURN.
```

292019-38

```
;***** TRANSMIT SUBROUTINE *****
```

```
TRAN:  MOV DPTR,#200H
        MOV R5,#0FFH
        MOV A,#0A8H
        CJNE A,SIUST,WAIT
        MOVX A,@DPTR
        INC DPTR
        DJNZ R5,NXTI
        MOV SIUST,#57H
        RET
NXTI:  MOV SIUST,#57H
        MOV A,#0B0H
        JMP WAIT
; EXT. RAM TRANSMIT BUFFER START
; EXT. RAM TRANSMIT BUFFER LENGTH
; CONTROL STATE
; WAIT FOR CONTROL BYTE TRANSMISSION
; MOVE DATA FROM EXT. RAM TO ACC.
; MOVE DATA INTO INT. RAM AT @TBS
; INCREMENT POINTER
; IS IT THE LAST I-BYTE ?
; XMIT THE LAST I-BYTE
; RETURN.
; KEEP "BYP" IN CONTROL STATE
; DMA-LOOP STATE
; TRANSMIT THE NEXT BYTE
```

292019-39

*****RECEIVE INTERRUPT ROUTINE*****

```

REC:  CLR    TRO          ; TURN OFF COUNTER 0
      MOV    DPTR,#200H   ; EXT. RAM RECEIVE BUFFER START
      MOV    R5,#0FFH     ; EXT. RAM RECEIVE BUFFER LENGTH
      MOV    A,#08H       ; ADDRESS STATE
HOLD:  CJNE   A,SIUST,HOLD ; WAIT FOR ADDRESS BYTE
      MOV    SIUST,#0EFH  ; MOVE "BYP" INTO CONTROL STATE
      ; SKIP THE ADDRESS STATE
      MOV    A,#18H       ; PUSH-1 STATE
WAIT1: CJNE   A,SIUST,WAIT1; WAIT FOR THE ADDRESS BYTE
      MOV    A,RCB        ; MOVE THE RECEIVED ADDRESS BYTE TO ACC.
      CJNE   A,STAD,WAIT2 ; ADDRESS MATCHED?
      SJMP   WAIT3        ; YES.
WAIT2: MOV    RCB,#00010000B ; MOVE INFO. CONTROL BYTE TO RCB
      MOV    SIUST,#0CFH  ; MOVE "BYP" INTO BOV-LOOP STATE
      RETI               ; RETURN

;
WAIT3: MOV    SIUST,#0EFH  ; MOVE "BYP" INTO CONTROL STATE
      MOV    A,#18H       ; PUSH-1 STATE
WAIT4: CJNE   A,SIUST,WAIT4; WAIT FOR THE CONTROL BYTE
      MOV    A,RCB        ; MOVE RECEIVE CONTROL BYTE INTO ACC.
      JB     ACC.0,RTRN   ; IF NOT AN I-FRAME RETURN
      PUSH   RCB          ; SAVE RECEIVE CONTROL BYTE
NEXTI: MOV    SIUST,#0EFH  ; PUSH "BYP" INTO CONTROL STATE(10H).
      MOV    A,#18H       ; PUSH-1 STATE
WAIT5: CJNE   A,SIUST,WAIT5; WAIT FOR AN I-BYTE
      MOV    A,RCB        ; MOVE RECEIVED I-BYTE INTO ACC.
      MOVX   @DPTR,A      ; MOVE DATA TO EXT. RAM
      INC    DPTR         ; INCREMENT PTR TO EXTERNAL RAM
      DJNZ   R5,NEXTI     ; IS IT THE LAST I-BYTE?
      POP    RCB          ; YES. RESTORE THE CONTENTS OF RCB
RTRN:  RETI               ; RETURN
END

```

292019-40

98-210305

98-210305

98-210305

80186/188 Application Notes

5

5

80186\188 Application Notes

in the data sheet. The second as to describe, through examples, the use of the 80186 with other digital logic such as memory.

The 80186 family actually consists of 4 devices: the original 80186 and 80188, and the new 80C186 and 80C188 microprocessors manufactured on Intel's CHMOS III process. The 80186 and 80C186 are 16-bit microprocessors but have 8-bit external data buses. The 80C186 and 80C188 offer the advantage of increased speed (up to 16 MHz) and important new features including a Refresh Control Unit, Power-Save Logic, and ONCE™ Mode (see Figure 2). For simplicity, this Application Note uses the name 80186 to refer collectively to all the members of the 80186 family. Differences between individual processors are pointed out as necessary.

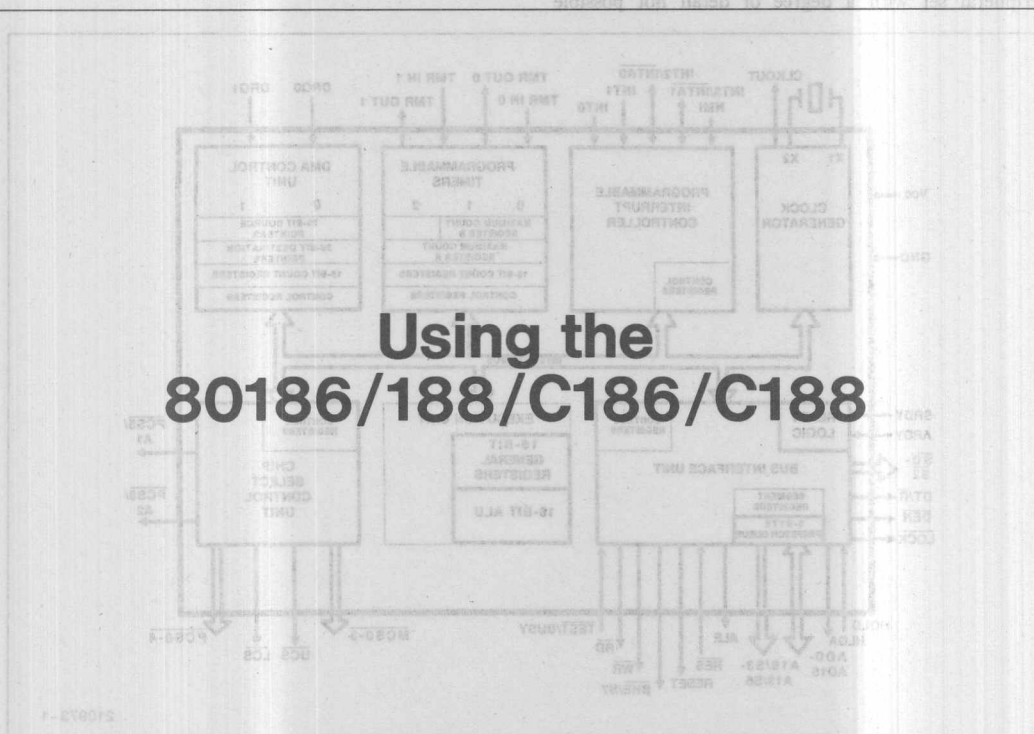
1.0 INTRODUCTION

The 80186 microprocessor family holds the position of industry standard among high integration microprocessors. VLSI technology incorporates the most commonly used peripheral functions with a 16-bit CPU on the same silicon die to ensure compatibility and high reliability (see Figure 1). The 80186 reputation for flexibility and uncomplicated programming make it the first choice microprocessor for such data control applications as local area network equipment, PC add-on cards, terminals, disk storage subsystems, avionics, and medical instrumentation.

There are two pointers to this Application Note. The first is the operation of the integrated 80186 device of detail not possible.

October 1988

Using the 80186/188/C186/C188



1.0 INTRODUCTION

The 80186 microprocessor family holds the position of industry standard among high integration microprocessors. VLSI technology incorporates the most commonly used peripheral functions with a 16-bit CPU on the same silicon die to assure compatibility and high reliability (see Figure 1). The 80186 reputation for flexibility and uncomplicated programming make it the first choice microprocessor for such data control applications as local area network equipment, PC add-on cards, terminals, disk storage subsystems, avionics, and medical instrumentation.

There are two purposes to this Application Note. The first is to explain the operation of the integrated 80186 peripheral set with a degree of detail not possible

in the data sheet. The second is to describe, through examples, the use of the 80186 with other digital logic such as memory.

The 80186 family actually consists of 4 devices: the original 80186 and 80188, and the new 80C186 and 80C188 microprocessors manufactured on Intel's CHMOS III process. The 80188 and 80C188 are 16-bit microprocessors but have 8-bit external data buses. The 80C186 and 80C188 offer the advantage of increased speed (up to 16 MHz) and important new features including a Refresh Control Unit, Power-Save Logic, and ONCE™ Mode (see Figure 2). For simplicity, this Ap Note uses the name 80186 to refer collectively to all the members of the 80186 family. Differences between individual processors are pointed out as necessary.

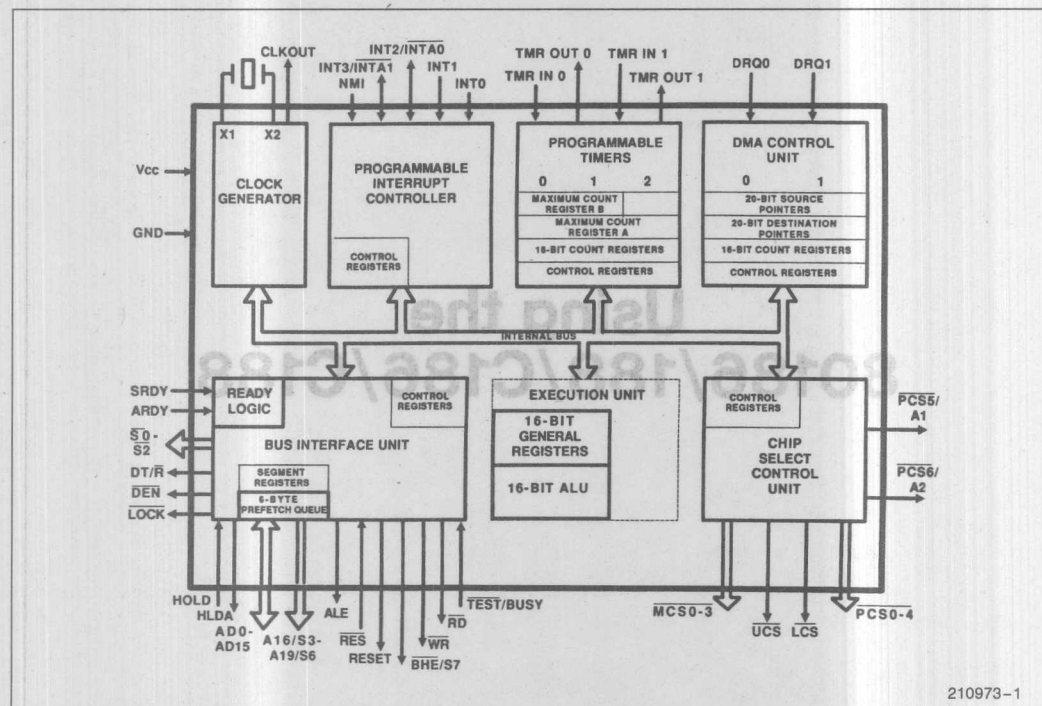


Figure 1. 80186 Block Diagram

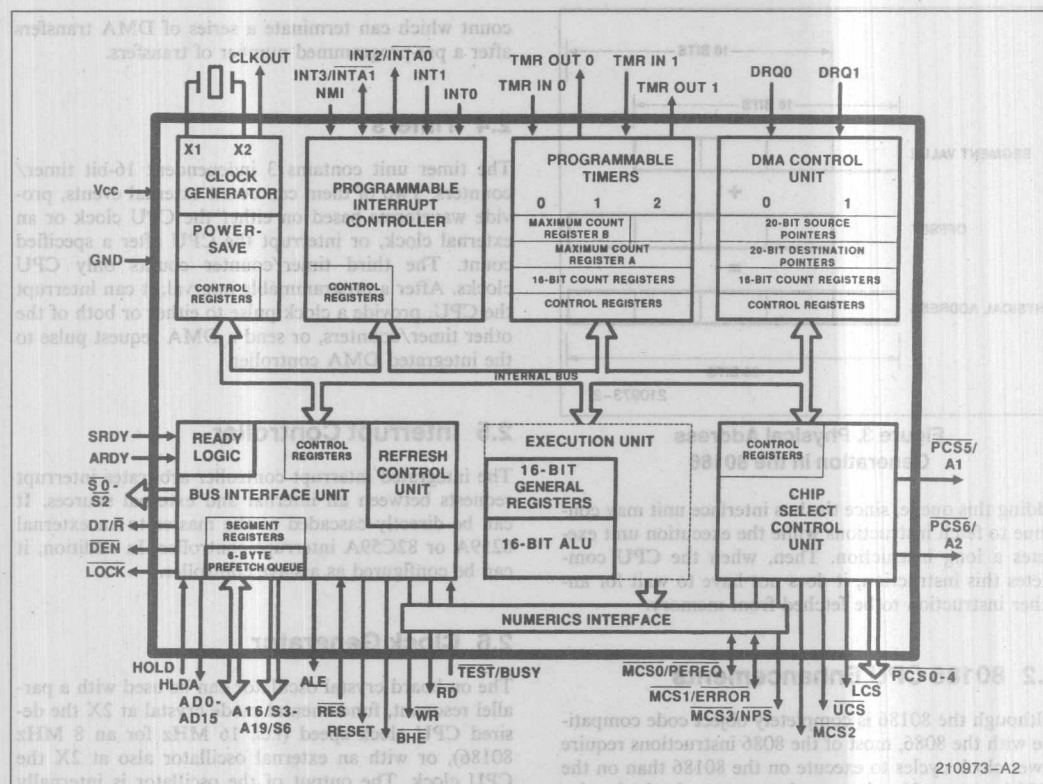


Figure 2. 80C186 Block Diagram

2.0 OVERVIEW OF THE 80186 FAMILY

2.1 The CPU

The 80186 CPU shares a common base architecture with the 8086, 8088, 80286, and 80386 processors. It is completely object code compatible with the 8086/88. This architecture features four 16-bit general purpose registers (AX, BX, CX, DX) which may be used as operands in most arithmetic operations in either 8- or 16-bit units. It also features four 16-bit pointer registers (SI, DI, BP, SP) which may be used both in arithmetic operations and in accessing memory based variables. Four 16-bit segment registers (CS, DS, SS, ES) allow simple memory partitioning to aid construction of modular programs. Finally, it has a 16-bit instruction pointer and a 16-bit status register.

Physical memory addresses are generated by the 80186 identically to the 8086. The 16-bit segment value is shifted left 4 bits and then added to an offset value which is derived from combinations of the pointer

registers, the instruction pointer, and immediate values (see Figure 3). Any carry of this addition is ignored. The result is a 20-bit physical address.

The 80186 has a 16-bit ALU which performs 8 or 16-bit arithmetic and logical operations. It provides for data movement among registers, memory and I/O space. In addition, the CPU allows for high speed data transfer from one area of memory to another using string move instructions, and to or from an I/O port and memory using block I/O instructions. Finally, the CPU provides many conditional branch and control instructions.

In the 80186, as in the 8086, instruction fetching and instruction execution are performed by separate units: the bus interface unit and the execution unit, respectively. The 80186 also has a 6-byte prefetch queue as does the 8086. The 80188 has a 4-byte prefetch queue as does the 8088. As a program is executing, opcodes are fetched from memory by the bus interface unit and placed in this queue. Whenever the execution unit requires another opcode byte, it takes the byte out of the queue. Effective processor throughput is increased by

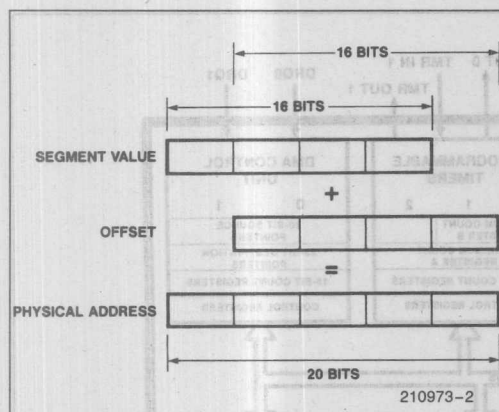


Figure 3. Physical Address Generation in the 80186

adding this queue, since the bus interface unit may continue to fetch instructions while the execution unit executes a long instruction. Then, when the CPU completes this instruction, it does not have to wait for another instruction to be fetched from memory.

2.2 80186 CPU Enhancements

Although the 80186 is completely object code compatible with the 8086, most of the 8086 instructions require fewer clock cycles to execute on the 80186 than on the 8086 because of hardware enhancements in the bus interface unit and the execution unit. In addition, the 80186 has many new instructions which simplify assembly language programming, enhance the performance of high level language implementations, and reduce code size. The added instructions are described in Appendix H of this Ap Note.

2.3 DMA Unit

The 80186 includes a DMA unit which provides two flexible DMA channels. This DMA unit will perform transfers to or from any combination of I/O space and memory space in either byte or word units. Every DMA cycle requires two to four bus cycles, one or two to fetch the data and one or two to deposit the data. This allows word data to be located on odd boundaries, or byte data to be moved from odd locations to even locations.

Each DMA channel maintains independent 20-bit source and destination pointers. Each of these pointers may independently address either I/O or memory space. After each DMA cycle, the pointers may be optionally incremented, decremented, or maintained constant. Each DMA channel also maintains a transfer

count which can terminate a series of DMA transfers after a pre-programmed number of transfers.

2.4 Timers

The timer unit contains 3 independent 16-bit timer/counters. Two of them can count external events, provide waveforms based on either the CPU clock or an external clock, or interrupt the CPU after a specified count. The third timer/counter counts only CPU clocks. After a programmable interval, it can interrupt the CPU, provide a clock pulse to either or both of the other timer/counters, or send a DMA request pulse to the integrated DMA controller.

2.5 Interrupt Controller

The integrated interrupt controller arbitrates interrupt requests between all internal and external sources. It can be directly cascaded as the master to an external 8259A or 82C59A interrupt controller. In addition, it can be configured as a slave controller.

2.6 Clock Generator

The on-board crystal oscillator can be used with a parallel resonant, fundamental mode crystal at 2X the desired CPU clock speed (i.e., 16 MHz for an 8 MHz 80186), or with an external oscillator also at 2X the CPU clock. The output of the oscillator is internally divided by two to provide the 50% duty cycle CPU clock from which all 80186 system timing is derived. The CPU clock is externally available, and all timing parameters are referenced to it.

2.7 Chip Select and Ready Generation Unit

The 80186 includes integrated chip select logic which can be used to enable memory or peripheral devices. Six output lines are used for memory addressing and seven output lines are used for peripheral addressing.

The memory chip select lines are split into 3 groups for separately addressing the major memory areas in a typical 80186 system: upper memory for reset ROM, lower memory for interrupt vectors, and mid-range memory for program memory. The size of each of these regions is user programmable. The starting location and ending location of lower memory and upper memory are fixed at 00000H and FFFFFH respectively; the starting location of the mid-range memory is user programmable.

Each of the seven peripheral select lines address one of seven contiguous 128 byte blocks above a programmable base address. This base address can be located in

either memory or I/O space so that peripheral devices may be I/O or memory mapped.

Each of the programmed chip select areas has associated with it a set of programmable ready bits. These bits allow a programmable number of wait states (0 to 3) to be automatically inserted whenever an access is made to the area of memory associated with the chip select area. In addition, a bit determines whether the external ready signals (ARDY and SRDY) will be used, or whether they will be ignored (i.e., the bus cycle will terminate even though a ready has not been returned on the external pins). There are 5 total sets of ready bits which allow independent ready generation for each of upper memory, lower memory, mid-range memory, peripheral devices 0-3 and peripheral devices 4-6.

2.8 Integrated Peripheral Accessing

The integrated peripheral and chip select circuitry is controlled by sets of 16-bit registers accessed using standard input, output, or memory access instructions. These peripheral control registers are all located within a 256 byte block which can be placed in either memory or I/O space. Because they are accessed exactly as if they were external devices, no new instruction types are required to access and control the integrated peripherals.

3.0 USING THE 80186 FAMILY

3.1 Bus Interfacing to the 80186

3.1.1 OVERVIEW

The 80186 bus structure is very similar to that of the 8086. It includes a multiplexed address/data bus, along with various control and status lines (see Table 1). Each bus cycle requires a minimum of 4 CPU clock cycles along with any number of wait states required to accommodate access limitations of external memory or peripheral devices. The bus cycles initiated by the 80186 CPU are identical to the bus cycles initiated by the 80186 integrated DMA unit.

Each clock cycle of the 80186 bus cycle is called a "T" state, and are numbered sequentially T_1 , T_2 , T_3 , T_4 and T_1 . Additional idle T states (T_i) can occur between T_4 and T_1 when the processor requires no bus activity (instruction fetches, memory writes, I/O reads, etc.). The ready signals control the number of wait states (t_w) inserted in each bus cycle. The maximum number of wait states is unbounded.

The beginning of a T state is signaled by a high to low transition of the CPU clock. Each T state is divided into two phases, phase 1 (or the low phase) and phase 2 (or the high phase) (see Figure 4).

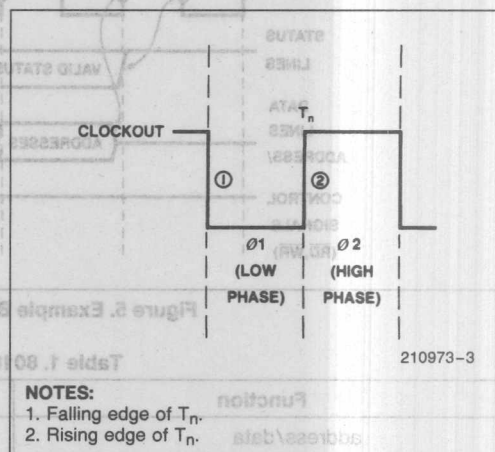


Figure 4. T-state in the 80186

Different types of bus activity occur for all of the T-states (see Figure 5). Address generation information occurs during T_1 , data generation during T_2 , T_3 , T_4 and T_1 . The beginning of a bus cycle is signaled by the status lines of the processor going from a passive state (all high) to an active state in the middle of the T-state immediately before T_1 (either a T_4 or a T_i). Information concerning an impending bus cycle appears during the T-state immediately before the first T-state of the cycle itself. Two different types of T_4 and T_i can be generated: one where the T state is immediately followed by a bus cycle, and one where the T state is immediately followed by an idle T state.

During the first type of T_4 or T_i , status information concerning the impending bus cycle is generated for the bus cycle immediately to follow. This information will be available no later than t_{CHSV} after the low-to-high transition of the 80186 clock in the middle of the T state. During the second type of T_4 or T_i , the status outputs remain inactive because no bus cycle will follow. The decision on which type T_4 or T_i state to present is made at the beginning of the T-state preceding the T_4 or T_i state (see Figure 6). This determination has an effect on bus latency (see Section 3.3.2).

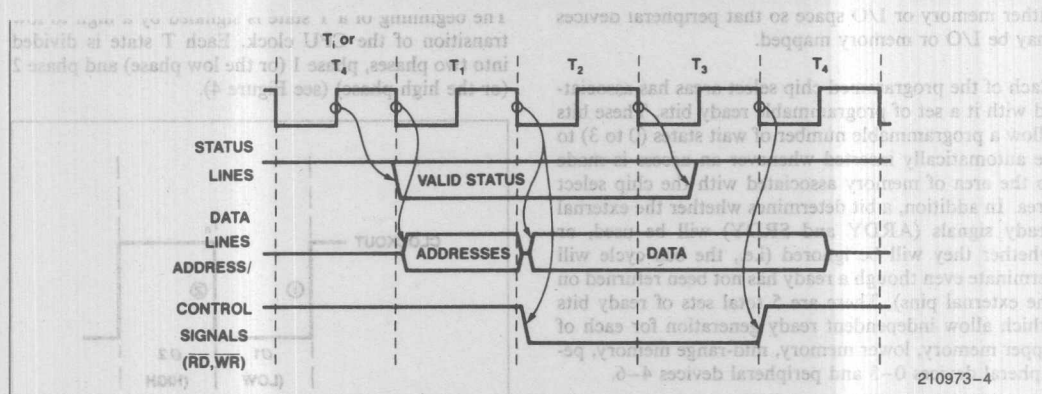


Figure 5. Example Bus Cycle of the 80186

Table 1. 80186 Bus Signals

Function	Signal Name
address/data	AD0-AD15
address/status	A16/S3-A19-S6, BHE/S7
co-processor control	TEST
local bus arbitration	HOLD, HLDA
local bus control	ALE, RD, WR, DT/R, DEN
multi-master bus	LOCK
ready (wait) interface	SRDY, ARDY
status information	S0-S2

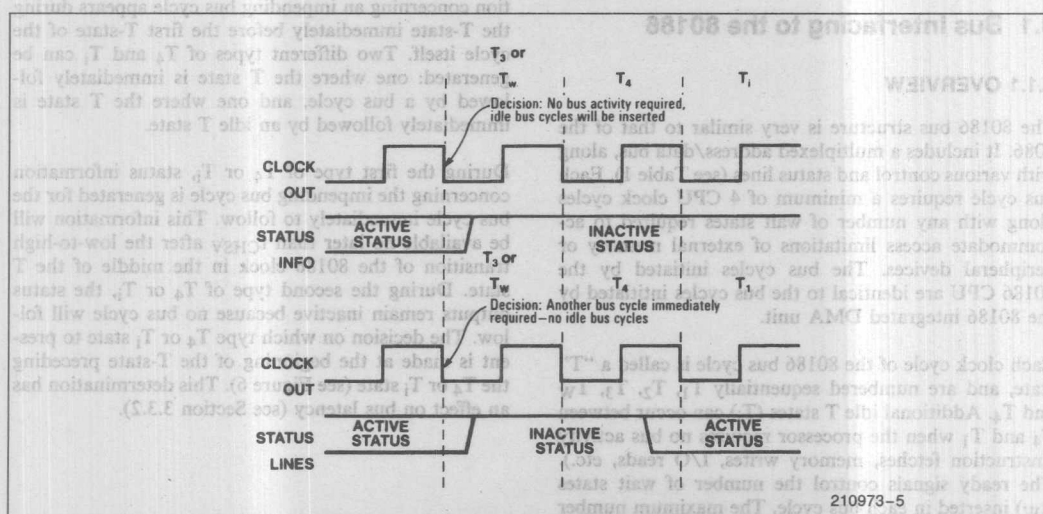


Figure 6. Active-Inactive Status Transitions in the 80186

3.1.2 PHYSICAL ADDRESS GENERATION

Physical addresses are generated by the 80186 during T_1 of a bus cycle. Since the address and data lines are multiplexed on the same set of pins, addresses must be latched during T_1 if they are required to remain stable for the duration of the bus cycle. To facilitate latching of the physical address, the 80186 generates an active high ALE (Address Latch Enable) signal which can be directly connected to a transparent latch's strobe input.

Figure 7 illustrates the physical address generation parameters of the 80186. Addresses are guaranteed valid no greater than t_{CLAV} after the beginning of T_1 , and remain valid at least t_{CLAX} after the end of T_1 . The ALE signal is driven high in the middle of the T state (either T_4 or T_5) immediately preceding T_1 and is driven low in the middle of T_1 , no sooner than t_{AVLL} after addresses become valid. This parameter (t_{AVLL}) is required to satisfy the address latch set-up times of address valid until strobe inactive. Addresses remain stable on the address/data bus at least t_{LLAX} after ALE goes inactive to satisfy address latch hold times.

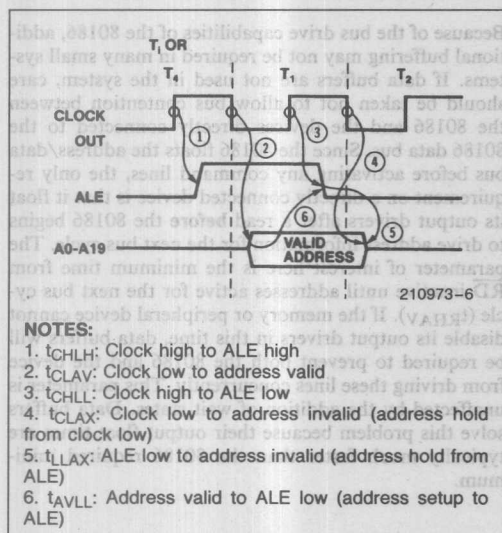


Figure 7. Address Generation Timing of the 80186

Because ALE goes high before addresses become valid, the delay through the address latches will be the propagation delay through the latch rather than the delay from the latch strobe, which is typically longer than the

propagation delay. Note that the 80186 drives ALE high one full clock phase earlier than the 8086 or the 82C88 bus controller, and keeps it high throughout the 8086 or 82C88 ALE high time (i.e., the 80186 ALE pulse is wider).

A typical circuit for latching physical addresses is shown in Figure 8. This circuit uses 3 transparent octal non-inverting latches to demultiplex all 20 address bits provided by the 80186/80188. Typically, the upper 4 address bits only select among various memory components or subsystems, so when the integrated chip selects (see Section 8) are used, these upper bits need not be latched. The worst case address generation time from the beginning of T_1 (including address latch propagation time for the circuit is:

$$t_{CLAV} + t_{PD}$$

Many memory or peripheral devices may not require addresses to remain stable throughout a data transfer. If a system is constructed wholly with these types of devices, addresses need not be latched. In addition, two of the peripheral chip select outputs of the 80186 may be configured to provide latched A1 and A2 outputs for peripheral register selects in a system which does not demultiplex the address/data bus.

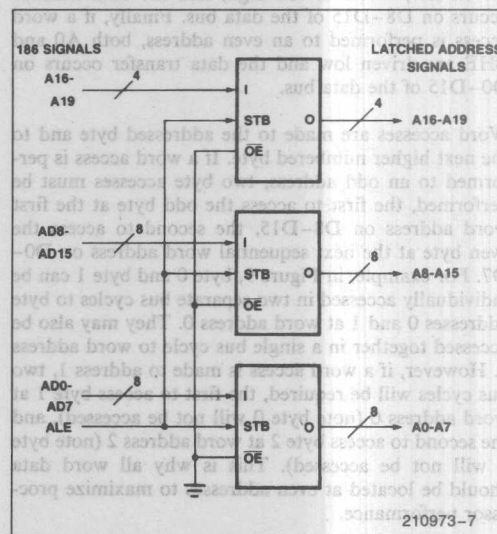


Figure 8. Demultiplexing the Address Bus of the 80186 Using Transparent Latches

One more signal is generated by the 80186 to address memory: BHE (Bus High Enable). This signal, along with A0, is used to enable byte devices connected to either or both halves (bytes) of the 16-bit data bus. Because A0 is used only to enable devices onto the lower half of the data bus, memory chip address inputs are usually driven by address bits A1–A19, not A0–A19. This provides 512K unique word addresses, or 1M unique byte addresses. BHE is not present on the 8-bit 80188. All data transfers occur on the 8-bits of the data bus.

3.1.3 80186/80C186 DATA BUS OPERATION

Throughout T₂, T₃, T_W and T₄ of a bus cycle the multiplexed address/data bus becomes a 16-bit data bus. Data transfers on this bus may be either bytes or words. All memory is byte addressable (see Figure 9).

All bytes with even addresses (A0 = 0) reside on the lower 8 bits of the data bus, while all bytes with odd addresses (A0 = 1) reside on the upper 8 bits of the data bus. Whenever an access is made to only the even byte, A0 is driven low, BHE is driven high, and the data transfer occurs on D0–D7 of the data bus. Whenever an access is made to only the odd byte, BHE is driven low, A0 is driven high, and the data transfer occurs on D8–D15 of the data bus. Finally, if a word access is performed to an even address, both A0 and BHE are driven low and the data transfer occurs on D0–D15 of the data bus.

Word accesses are made to the addressed byte and to the next higher numbered byte. If a word access is performed to an odd address, two byte accesses must be performed, the first to access the odd byte at the first word address on D8–D15, the second to access the even byte at the next sequential word address on D0–D7. For example, in Figure 9, byte 0 and byte 1 can be individually accessed in two separate bus cycles to byte addresses 0 and 1 at word address 0. They may also be accessed together in a single bus cycle to word address 0. However, if a word access is made to address 1, two bus cycles will be required, the first to access byte 1 at word address 0 (note byte 0 will not be accessed), and the second to access byte 2 at word address 2 (note byte 3 will not be accessed). This is why all word data should be located at even addresses to maximize processor performance.

When byte reads are made, the data returned on the unused half of the data bus is ignored. When byte writes are made, the data driven on the unused half of the data bus is indeterminate.

3.1.4 80188/80C188 DATA BUS OPERATION

Because the 80188 and 80C188 externally have only 8-bit data buses, the above discussion about upper and lower bytes of the data bus does not apply. No performance improvement will occur if word data is placed on even boundaries in memory space. All word accesses require two bus cycles, the first to access to lower byte of the word; the second to access the upper byte of the word.

Any 80188/80C188 access to the integrated peripherals is performed 16 bits at a time, whether byte or word addressing is used. If a byte operation is used, the external bus only indicates a single byte transfer even though the word access takes place.

3.1.5 GENERAL DATA BUS OPERATION

Because of the bus drive capabilities of the 80186, additional buffering may not be required in many small systems. If data buffers are not used in the system, care should be taken not to allow bus contention between the 80186 and the devices directly connected to the 80186 data bus. Since the 80186 floats the address/data bus before activating any command lines, the only requirement on a directly connected device is that it float its output drivers after a read before the 80186 begins to drive address information for the next bus cycle. The parameter of interest here is the minimum time from RD inactive until addresses active for the next bus cycle (t_{RHAY}). If the memory or peripheral device cannot disable its output drivers in this time, data buffers will be required to prevent both the 80186 and the device from driving these lines concurrently. This parameter is unaffected by the addition of wait states. Data buffers solve this problem because their output float times are typically much faster than the 80186 required minimum.

Figure 7. Address Generation Timing of the 80186

Because ALE goes high before addresses become valid, the delay through the address latches will be the propagation delay through the latch rather than the delay from the latch strobe, which is typically longer than the

Figure 8. Demultiplexing the Address Bus of the 80188 Using Transparent Latches

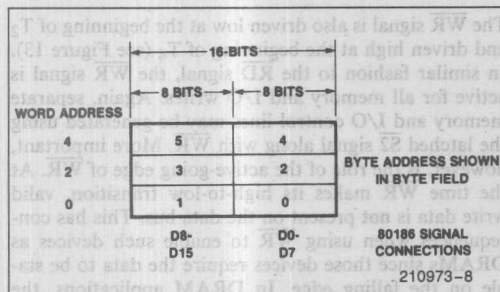


Figure 9. Physical Memory Byte/Word Addressing in the 80186

If data buffers are required, the 80186 provides $\overline{\text{DEN}}$ (Data ENable) and $\text{DT}/\overline{\text{R}}$ (Data Transmit/Receive) signals to simplify buffer interfacing. The $\overline{\text{DEN}}$ and $\text{DT}/\overline{\text{R}}$ signals are activated during all bus cycles. The $\overline{\text{DEN}}$ signal is driven low whenever the processor is either ready to receive data (during a read) or when the processor is ready to send data (during a write). In other words, $\overline{\text{DEN}}$ is low during any active bus cycle when address information is not being generated on the address/data pins. In most systems, the $\overline{\text{DEN}}$ signal should not be directly connected to the OE input of buffers, since unbuffered devices (or other buffers) may be directly connected to the processor's address/data pins. If $\overline{\text{DEN}}$ were directly connected to several buffers, contention would occur during read cycles, as many devices attempt to drive the processor bus. Rather, it should be a factor (along with the chip selects for buffered devices) in generating the output enable.

The $\text{DT}/\overline{\text{R}}$ signal determines the direction of data through the bi-directional buffers. It is high whenever

data is being written from the processor, and is low whenever data is being read into the processor. Unlike the $\overline{\text{DEN}}$ signal, it may be directly connected to bus buffers, since this signal does not usually enable the output drivers of the buffer. An example data bus subsystem supporting both buffered and unbuffered devices is shown in Figure 10. Note that the A side of the buffer is connected to the 80186, the B side to the external device. The $\text{DT}/\overline{\text{R}}$ signal can directly drive the T (transmit) signal of a typical buffer since it has the correct polarity.

3.1.6 CONTROL SIGNALS

The 80186 directly provides the control signals $\overline{\text{RD}}$, $\overline{\text{WR}}$, $\overline{\text{LOCK}}$ and $\overline{\text{TEST}}$. In addition, the 80186 provides the status signals S0-S2 and S6 from which all other required bus control signals can be generated.

3.1.6.1 $\overline{\text{RD}}$ and $\overline{\text{WR}}$

The $\overline{\text{RD}}$ and $\overline{\text{WR}}$ signals strobe data to or from memory or I/O space. The $\overline{\text{RD}}$ signal is driven low at the beginning of T_2 , and is driven high at the beginning of T_4 during all memory and I/O reads (see Figure 11). $\overline{\text{RD}}$ will not become active until the 80186 has ceased driving address information on the address/data bus. Data is sampled into the processor at the beginning of T_4 . $\overline{\text{RD}}$ will not go inactive until the processor's data hold time has been satisfied.

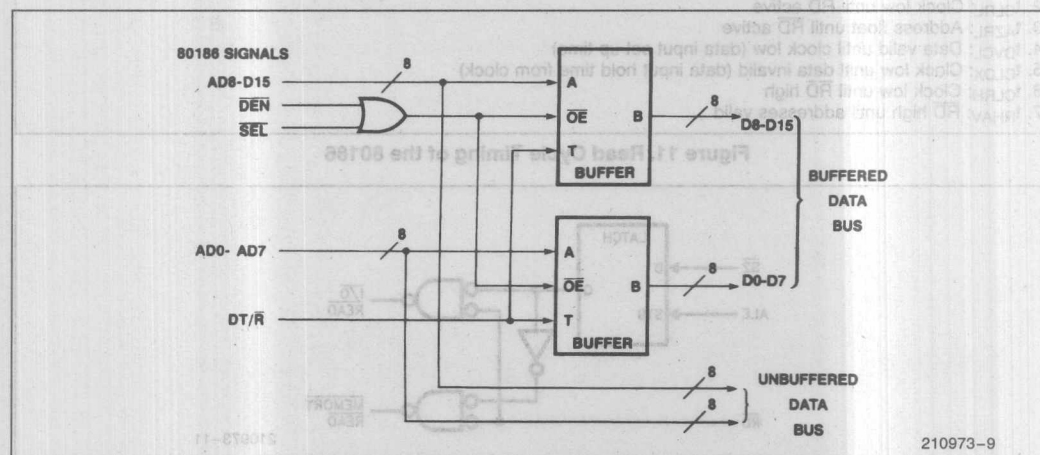


Figure 10. Example 80186 Buffered/Unbuffered Data Bus

Note that the 80186 does not provide separate I/O and memory \overline{RD} signals. If separate I/O read and memory read signals are required, they can be synthesized using the $\overline{S2}$ signal (which is low for all I/O operations and high for all memory operations) and the \overline{RD} signal (see Figure 12). It should be noted that if this approach is used, the $\overline{S2}$ signal will require latching, since the $\overline{S2}$ signal (like $\overline{S0}$ and $\overline{S1}$) goes to an inactive state well before the beginning of T_4 (where \overline{RD} goes inactive). If $\overline{S2}$ was directly used for this purpose, the type of read command (I/O or memory) could change just before T_4 as $\overline{S2}$ goes to the inactive state (high). The status signals may be latched using ALE the same as the address signals (often using the spare bits in the address latches).

Often the lack of a separate I/O and memory \overline{RD} signal is not important in an 80186 system. Each 80186 chip select signal will respond to accesses exclusively in memory or I/O space. Thus, when a chip select is used, the external device is enabled only during accesses to the proper address in the proper space.

The \overline{WR} signal is also driven low at the beginning of T_2 and driven high at the beginning of T_4 (see Figure 13). In similar fashion to the \overline{RD} signal, the \overline{WR} signal is active for all memory and I/O writes. Again, separate memory and I/O control lines may be generated using the latched $\overline{S2}$ signal along with \overline{WR} . More important, however, is the role of the active-going edge of \overline{WR} . At the time \overline{WR} makes its high-to-low transition, valid write data is not present on the data bus. This has consequences when using \overline{WR} to enable such devices as DRAMs since those devices require the data to be stable on the falling edge. In DRAM applications, the problem is solved by the DRAM controller (an Intel 8207, for example). For other applications which require valid data before the \overline{WR} transition, place cross-coupled NAND gates between the CPU and the device on the \overline{WR} line (see Figure 14). The added gates delay the active-going edge of \overline{WR} to the device by one clock phase, at which time valid data is driven on the bus by the 80186.

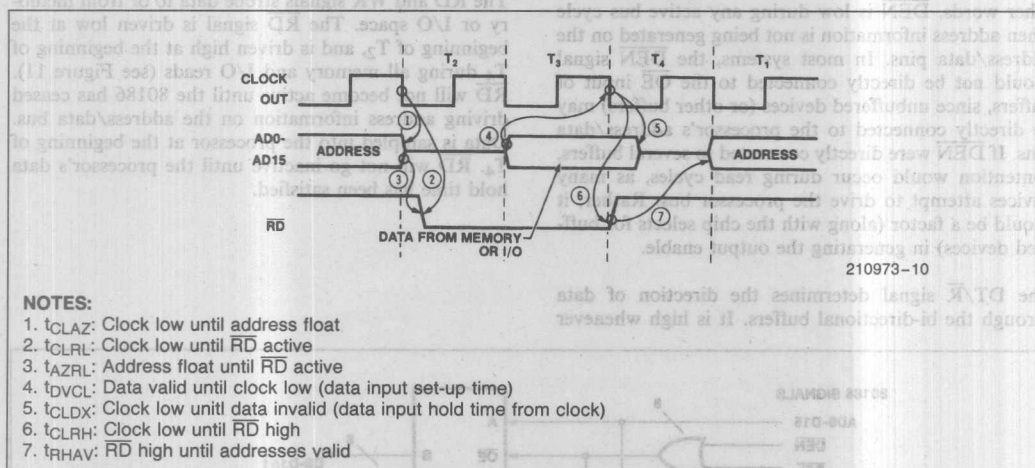


Figure 11. Read Cycle Timing of the 80186

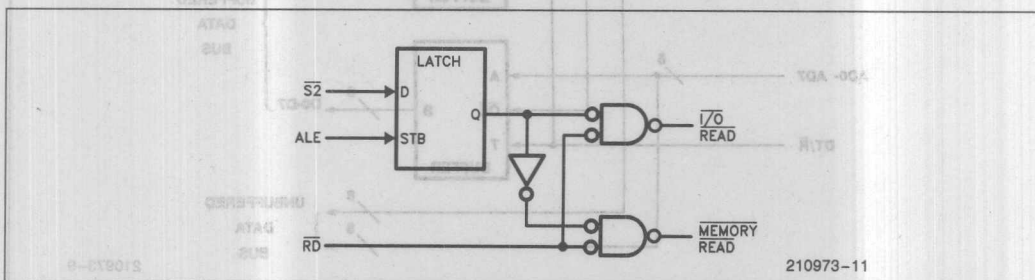


Figure 12. Generating I/O and Memory Read Signals from the 80186

3.1.6.2 Queue Status Signals

If the RD line is externally grounded during reset and remains grounded during processor operation, the 80186 will enter Queue Status Mode. When in this mode, the WR and ALE signals become queue status outputs, reflecting the status of the internal prefetch queue during each clock cycle. These signals are provided to allow a processor extension (such as the Intel 8087 floating point processor) to track execution of instructions within the 80186. The interpretation of QS0 (ALE) and QS1 (WR) is given in Table 2. These signals change on the high-to-low clock transition, one clock phase earlier than on the 8086. Note that since execution unit operation is independent of bus interface unit operation, queue status lines may change in any T state.

Table 2. 80186 Queue Status

QS1	QS0	Interpretation
0	0	no operation
0	1	first byte of instruction taken from queue
1	0	queue was reinitialized
1	1	subsequent byte of instruction taken from queue

Since the ALE, RD, and WR signals are not directly available from the 80186 when it is configured in queue status mode, these signals must be derived from the status lines S0-S2 using an external 82C88 bus controller (see Figure 15). To prevent the 80186 from accidentally entering queue status mode during reset, the RD line is internally provided with a weak pullup device.

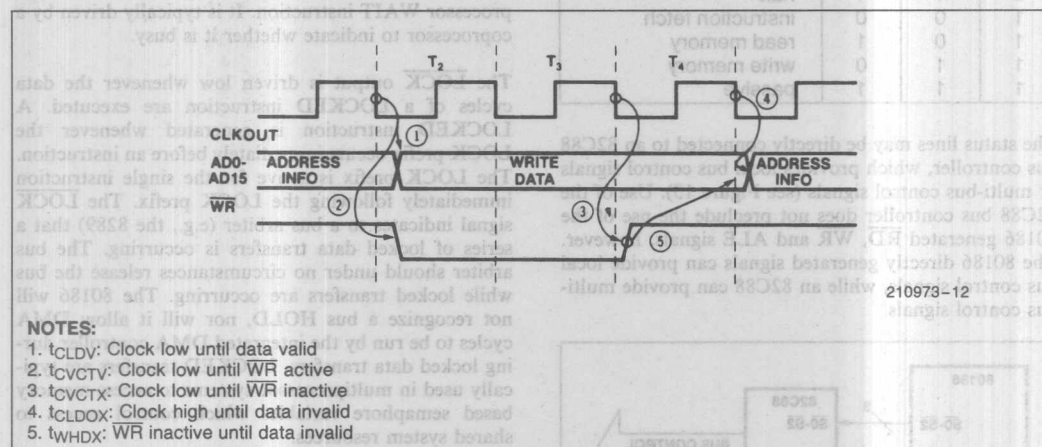


Figure 13. Write Cycle Timing of the 80186

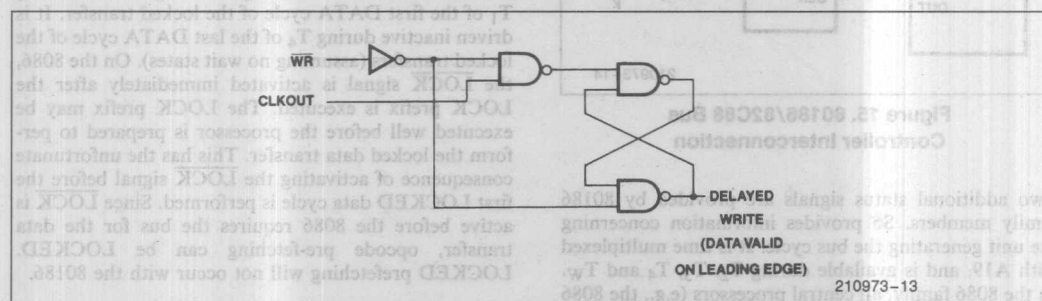


Figure 14. Synthesizing Delayed Write from the 80186

3.1.6.3 Status Lines

The 80186 provides 3 status outputs which indicate the type of bus cycle currently being executed. These signals go from an inactive state (all high) to one of seven possible active states during the T state immediately preceding T₁ of a bus cycle (see Figure 6). The possible status line encodings are given in Table 3. The status lines are driven to their inactive state in the T₃ or T_W state immediately preceding T₄ of the current bus cycle.

Table 3. 80186 Status Line Interpretation

S2	S1	S0	Operation
0	0	0	interrupt acknowledge
0	0	1	read I/O
0	1	0	write I/O
0	1	1	halt
1	0	0	instruction fetch
1	0	1	read memory
1	1	0	write memory
1	1	1	passive

The status lines may be directly connected to an 82C88 bus controller, which provides local bus control signals or multi-bus control signals (see Figure 15). Use of the 82C88 bus controller does not preclude the use of the 80186 generated RD, WR and ALE signals, however. The 80186 directly generated signals can provide local bus control signals, while an 82C88 can provide multi-bus control signals.

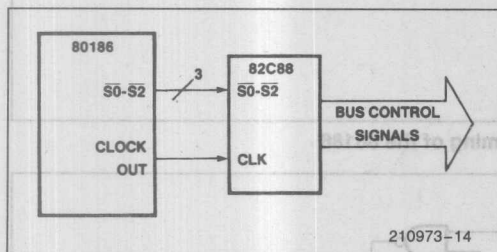


Figure 15. 80186/82C88 Bus Controller Interconnection

Two additional status signals are provided by 80186 family members. S6 provides information concerning the unit generating the bus cycle. It is time multiplexed with A19, and is available during T₂, T₃, T₄ and T_W. In the 8086 family, all central processors (e.g., the 8086 and 8087) drive this line low, while all I/O processors (e.g., 8089) drive this line high during their respective bus cycles. Following this scheme, the 80186 drives this line low whenever the bus cycle is generated by the 80186 CPU, but drives it high when the bus cycle is generated by the integrated 80186 DMA unit. This allows external devices to distinguish between bus cycles fetching data for the CPU from those transferring data for the DMA unit.

S7 and $\overline{\text{BHE}}$ are logically equivalent signals provided by the 80186 and the 80C186 (see Section 3.1.2). S7 is always high on the 80188 and 80C188 (except during 80C188 DRAM refresh cycles) which signifies the presence of an 8-bit data bus.

Three other status signals are available on the 8086 but not on the 80186. They are S3, S4, and S5. Taken together, S3 and S4 indicate the segment register from which the current physical address has been derived. S5 indicates the state of the interrupt flip-flop. On the 80186, these signals will always be low.

3.1.6.4 TEST and LOCK

Finally, the 80186 provides a $\overline{\text{TEST}}$ input and a $\overline{\text{LOCK}}$ output. The $\overline{\text{TEST}}$ input is used in conjunction with the processor WAIT instruction. It is typically driven by a coprocessor to indicate whether it is busy.

The $\overline{\text{LOCK}}$ output is driven low whenever the data cycles of a LOCKED instruction are executed. A LOCKED instruction is generated whenever the LOCK prefix occurs immediately before an instruction. The LOCK prefix is active for the single instruction immediately following the LOCK prefix. The LOCK signal indicates to a bus arbiter (e.g., the 8289) that a series of locked data transfers is occurring. The bus arbiter should under no circumstances release the bus while locked transfers are occurring. The 80186 will not recognize a bus HOLD, nor will it allow DMA cycles to be run by the integrated DMA controller during locked data transfers. LOCKED transfers are typically used in multiprocessor systems to access memory based semaphore variables which control access to shared system resources.

On the 80186, the $\overline{\text{LOCK}}$ signal will go active during T₁ of the first DATA cycle of the locked transfer. It is driven inactive during T₄ of the last DATA cycle of the locked transfers (assuming no wait states). On the 8086, the $\overline{\text{LOCK}}$ signal is activated immediately after the LOCK prefix is executed. The LOCK prefix may be executed well before the processor is prepared to perform the locked data transfer. This has the unfortunate consequence of activating the $\overline{\text{LOCK}}$ signal before the first LOCKED data cycle is performed. Since LOCK is active before the 8086 requires the bus for the data transfer, opcode pre-fetching can be LOCKED. LOCKED prefetching will not occur with the 80186.

The LOCK output is also driven low during interrupt acknowledge cycles when the integrated interrupt controller operates in Cascade or Slave Modes (see Sections 6.5.2 and 6.5.3). In these modes, the operation of the LOCK pin may be altered when an interrupt occurs

during execution of a software-LOCKED instruction. See Section 6.5.4 for a description of additional hardware necessary to block DMA and HOLD requests under such circumstances.

3.1.7 HALT TIMING

A HALT bus cycle signifies that the 80186 CPU has executed a HLT instruction. It differs from a normal bus cycle in two ways.

The first way a HALT bus cycle differs from a normal bus cycle is that neither \overline{RD} nor \overline{WR} will be driven active. Address and data information will not be driven by the processor, and no data will be returned. The second way a HALT bus cycle differs from a normal bus cycle is that the $\overline{S0}$ – $\overline{S2}$ status lines go to their inactive state (all high) during T_2 of the bus cycle, well before they go to their inactive state during a normal bus cycle.

Like a normal bus cycle, however, ALE is driven active. Since no valid address information is present, the information strobed into the address latches should be ignored. This ALE pulse can be used, however, to latch the HALT status from the $\overline{S0}$ – $\overline{S2}$ status lines.

The processor being halted does not interfere with the operation of any of the 80186 integrated peripheral units. This means that if a DMA transfer is pending while the processor is halted, the bus cycles associated with the transfer will run. In fact, DMA latency time will improve while the processor is halted because the DMA unit will not be contending with the processor for access to the 80186 (see section 4.4.1).

3.1.8 82C88 AND 8289 INTERFACING

The 82C88 and 8289 are the bus controller and multi-master bus arbitration devices used with the 8086. Because the 80186 bus is similar to the 8086 bus, they can be used with the 80186. Figure 16 shows an 80186 interconnection to these two devices.

The 82C88 bus controller generates control signals (\overline{RD} , \overline{WR} , ALE, $\overline{DT/R}$, \overline{DEN} , etc.) for an 8086 maximum mode system. It derives its information by decoding status lines $\overline{S0}$ – $\overline{S2}$ of the processor. Because the 80186 and the 8086 drive the same status information on these lines, the 80186 can be directly connected to the 82C88 just as in an 8086 system. Using the 82C88 with the 80186 does not prevent using the 80186 control signals. Many systems require both local bus control signals and system bus control signals. In this type of system, the 80186 lines could be used as the local signals, with the 82C88 lines used as the system signals. Note that in an 80186 system, the 82C88 generated ALE pulse occurs later than that of the 80186 itself. In many multimaster bus systems, the 82C88 ALE pulse

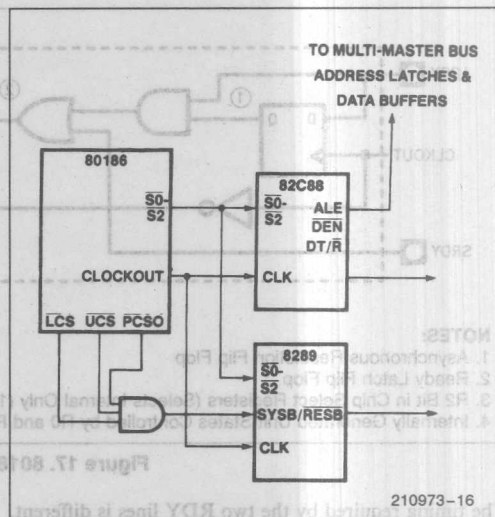


Figure 16. 80186/8288/8289 Interconnection

should be used to strobe the addresses into the system bus address latches to insure that the address hold times are met.

The 8289 bus arbiter arbitrates the use of a multi-master system bus among various devices, each of which can become the bus master. This component also decodes status lines $\overline{S0}$ – $\overline{S2}$ directly to determine when the system bus is required. When the system bus is required, the 8289 forces the processor to wait until it has acquired control of the bus, then it allows the processor to drive address, data and control information onto the system bus. The system determines when it requires system bus resources by an address decode. Whenever the address being driven coincides with the address of an on-board resource, the system bus is not required and thus will not be requested. The circuit shown in Figure 17 factors the 80186 chip select lines to determine when the system bus should be requested, or when the 80186 request can be satisfied using a local resource.

3.1.9 READY INTERFACING

The 80186 provides two ready lines, a synchronous ready (SRDY) line and an asynchronous ready (ARDY) line. These lines signal the bus controller to insert wait states (T_W) into a CPU bus cycle, allowing slower devices to respond to bus activity. Wait states will only be inserted when both ARDY and SRDY are low, i.e., only one of the lines need be active to terminate a bus cycle. Figure 17 depicts the logical ORING of the ARDY and SRDY functions. Any number of wait states may be inserted into a bus cycle. The 80186 will ignore the RDY inputs during any accesses to the integrated peripheral registers and to any area where the chip select ready bits indicate that the external ready should be ignored.



Figure 18 depicts activity for Normally-Ready and Normally-Not-Ready configurations of external logic. Remember that for ARDY to force wait states, SRDY must be low as well.

In a Normally-Ready implementation the setup and hold times of either the resolution flip-flop or the ready latch must be met. Wait states will be generated if ARDY goes inactive TARYHCH (also denoted



T_{ARYCH}) before the rising edge of T_2 and stays inactive a minimum of T_{ARYCHL} after the edge, or if $ARDY$ goes inactive at least T_{ARYLCL} before the falling edge of T_3 and stays inactive a minimum of T_{CLARX} after the edge. The 80186 ready circuitry performs this way to allow a slow device the maximum amount of time to respond with a not ready after it has been selected.

The synchronous ready (SRDY) line requires that all transitions on this line during T_2 , T_3 , or T_W satisfy setup and hold times (t_{SRYCL} and t_{CLSR} respectively). If these requirements are not met, the CPU will not function properly. Valid transitions on this line and subsequent wait state insertion is shown in Figure 19. The bus controller looks at SRDY at the beginning of each T_3 and T_W . If the line is sampled active at the beginning of either of these two cycles, that cycle will be immediately followed by T_4 . If the line is sampled inactive at the beginning of either T state, that cycle will be followed by a T_W . Any asynchronous transition on the SRDY line not occurring at the beginning of T_3 or T_W , i.e., when the processor is not sampling the input, will not cause CPU malfunction.

3.1.10 BUS PERFORMANCE ISSUES

Bus cycles occur sequentially, but do not necessarily come immediately one after another, that is the bus may remain idle for several T states (T_i) between each bus access initiated by the 80186. The reader should recall that a separate unit, the bus interface unit, fetches opcodes from memory, while the execution unit actually executes the pre-fetched instructions. The number of clock cycles required to execute an 80186 instruction vary from 2 clock cycles for a register to register move to 67 clock cycles for an integer divide.

If a program contains many long instructions, program execution will be CPU limited, that is, the instruction queue will be constantly filled. Thus, the execution unit does not need to wait for an instruction to be fetched. If a program contains mainly short instructions (for example, data move instructions), the execution will be bus limited. Here, the execution unit will have to wait often for an instruction to be fetched before it continues. Programs illustrating this effect and performance degradation of each with the addition of wait states are given in appendix G.

Although the amount of bus utilization will vary considerably from one program to another, a typical instruction mix on the 80186 will require greater bus utilization than the 8086. The 80186 executes most instructions in fewer clock cycles, thus requiring instructions from the queue at a faster rate. This also means that the effect of wait states is more pronounced in an 80186 system than in an 8086 system. In all but a few cases, however, the performance degradation incurred by adding a wait state is less than might be expected because instruction fetching and execution are performed by separate units.

3.2 Example Memory Systems

3.2.1 2764 INTERFACE

With the above knowledge of the 80186 bus, various memory interfaces may be generated. One of the simplest is the example EPROM interface shown in Figure 20.

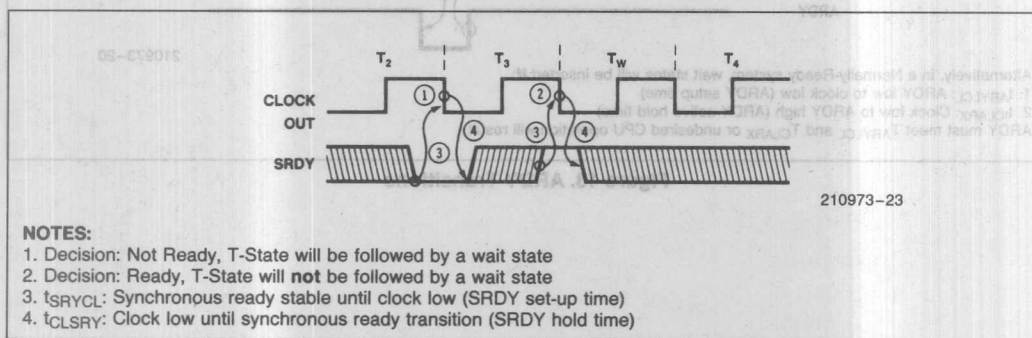


Figure 19. Valid SRDY Transitions



3.2.2 8203 DRAM INTERFACE

5-17

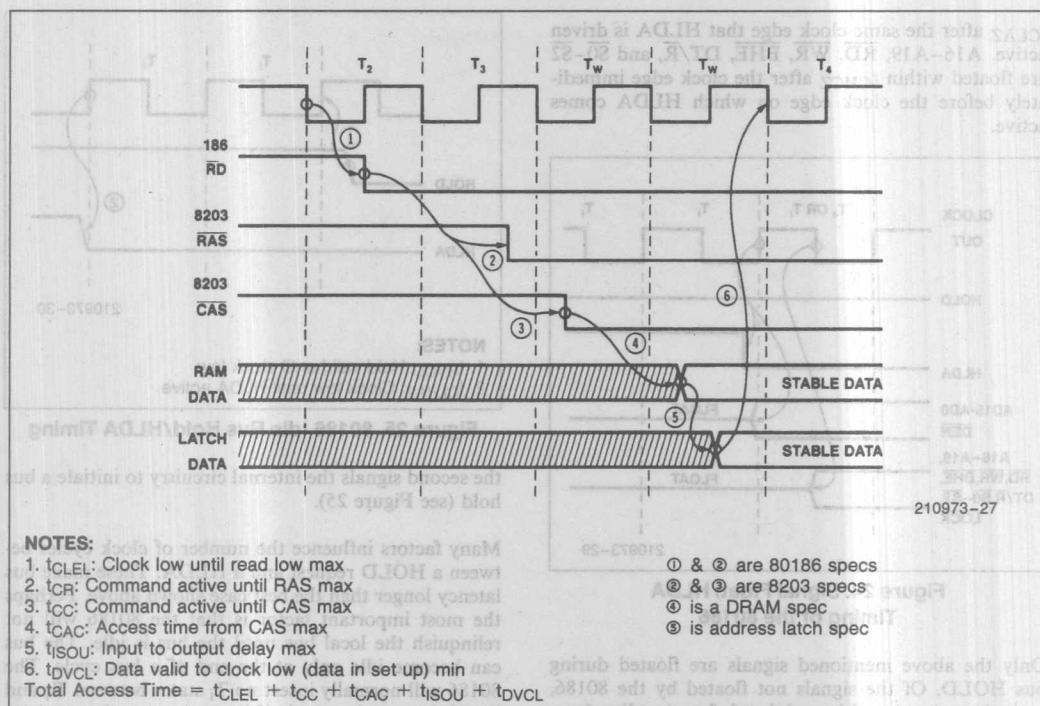


Figure 22. Example 8203 Access Time Calculation

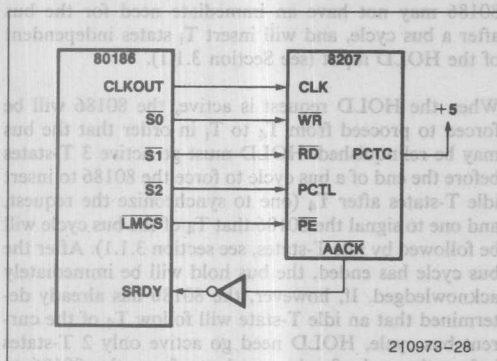


Figure 23. 80186/8207/DRAM Interface

cess cycles may occur simultaneously with an 8207 generated refresh cycle. When this occurs, the 8207 will hold the AACK line high until the processor initiated access is run (note, the sense of this line is reversed with respect to the 80186 SRDY input). This signal should be factored with the DRAM (8207) select input and used to drive the SRDY line of the 80186. Remember that either SRDY and ARDY needs to be active for a bus cycle to be terminated. If asynchronous devices (e.g., a Multibus interface) are connected to the ARDY line with the 8207 connected to the SRDY line,

care must be taken in design of the ready circuit such that only one of the RDY lines is driven active at a time to prevent premature termination of the bus cycle.

3.3 HOLD/HLDA Interface

The 80186 employs a HOLD/HLDA bus exchange protocol. This protocol allows other asynchronous bus masters (i.e., ones which drive address, data, and control information on the bus) to gain control of the bus.

3.3.1 HOLD RESPONSE

In the HOLD/HLDA protocol, a device requiring bus control (e.g., an external DMA device) raises the HOLD line. In response to this HOLD request, the 80186 will raise its HLDA line after it has finished its current bus activity. When the external device is finished with the bus, it drops its bus HOLD request. The 80186 responds by dropping its HLDA line and resuming bus operation.

When the 80186 recognizes a bus hold by driving HLDA high, it will float many of its signals (see Figure 24). AD0-AD15 and \overline{DEN} are floated within

t_{CLAZ} after the same clock edge that $HLDA$ is driven active. $A16-A19$, RD , WR , BHE , DT/R , and $S0-S2$ are floated within t_{CHCZ} after the clock edge immediately before the clock edge on which $HLDA$ comes active.

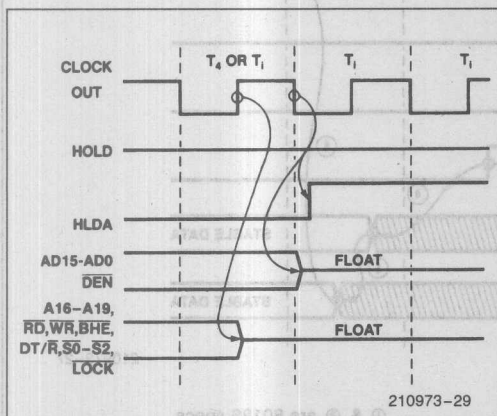


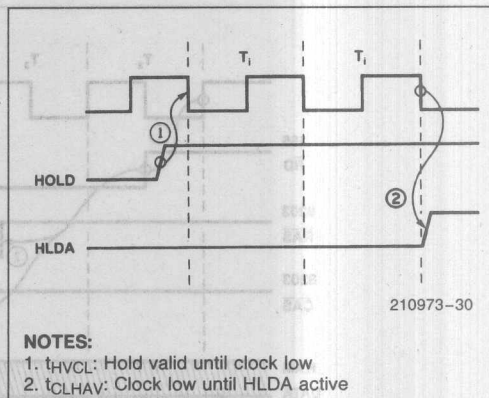
Figure 24. Signal Float/HLDA Timing of the 80186

Only the above mentioned signals are floated during bus HOLD. Of the signals not floated by the 80186, some have to do with peripheral functionality (e.g., TMR OUT). Many others either directly or indirectly control bus devices. These signals are ALE and all the chip select lines (UCS, LCS, MCS0-3, and PCS0-6).

3.3.2 HOLD/HLDA TIMING AND BUS LATENCY

The time required between HOLD going active and the 80186 driving HLDA active is known as bus latency. Many factors affect bus latency, including synchronization delays, bus cycle times, locked transfer times and interrupt acknowledge cycles.

The HOLD request line is internally synchronized by the 80186, and may therefore be an asynchronous input. To guarantee recognition on a particular clock edge, it must satisfy setup and hold times to the falling edge of the CPU clock. A full CPU clock cycle is required for synchronization (see Appendix B). If the bus is idle, HLDA will follow HOLD by two CPU clock cycles plus a small amount of setup and propagation delay time. The first clock cycle synchronizes the input;



NOTES:

1. t_{HVC} : Hold valid until clock low
2. t_{CLHAV} : Clock low until HLDA active

Figure 25. 80186 Idle Bus Hold/HLDA Timing

the second signals the internal circuitry to initiate a bus hold (see Figure 25).

Many factors influence the number of clock cycles between a HOLD request and a HLDA. These make bus latency longer than the best case shown above. Perhaps the most important factor is that the 80186 will not relinquish the local bus until the bus is idle. The bus can become idle only at the end of a bus cycle. The 80186 will normally insert no T_1 states between T_4 and T_1 of the next bus cycle if it requires any bus activity (e.g., instruction fetches or I/O reads). However, the 80186 may not have an immediate need for the bus after a bus cycle, and will insert T_1 states independent of the HOLD input (see Section 3.1.1).

When the HOLD request is active, the 80186 will be forced to proceed from T_4 to T_1 in order that the bus may be relinquished. HOLD must go active 3 T-states before the end of a bus cycle to force the 80186 to insert idle T-states after T_4 (one to synchronize the request, and one to signal the 80186 that T_4 of the bus cycle will be followed by idle T-states, see section 3.1.1). After the bus cycle has ended, the bus hold will be immediately acknowledged. If, however, the 80186 has already determined that an idle T-state will follow T_4 of the current bus cycle, HOLD need go active only 2 T-states before the end of a bus cycle to force the 80186 to relinquish the bus. This is because the external HOLD request is not required to force the generation of idle T-states. Figure 26 graphically portrays the scenarios depicted above.

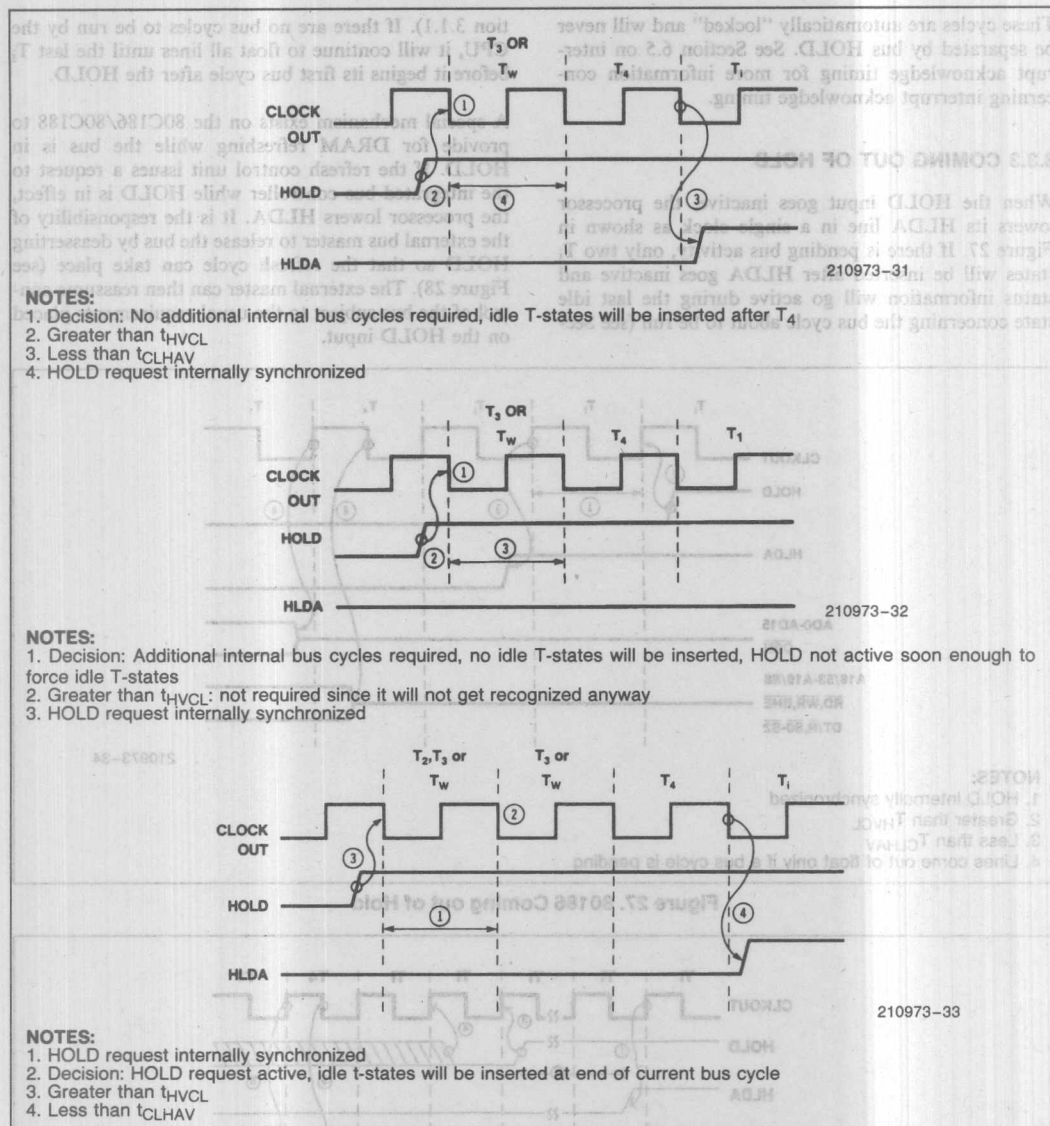


Figure 26. HOLD/HLDA Timing in the 80186

An external HOLD has higher priority than both the 80186 CPU or integrated DMA unit. However, an external HOLD will not separate the two cycles needed to perform a word access when the word accessed is located at an odd location (see Section 3.1.3). In addition, an external HOLD will not separate the two-to-four bus cycles required for the integrated DMA unit to perform a transfer. Each of these factors will add to the bus latency of the 80186.

Another factor influencing bus latency time is locked transfers. Whenever a locked transfer is occurring, the

80186 will not recognize external HOLDs (nor will it recognize internal DMA bus requests). Locked transfers are programmed by preceding an instruction with the LOCK prefix. String instructions may be locked. Since string transfers may require thousands of bus cycles, bus latency time will suffer if they are locked.

The final factor affecting bus latency time is interrupt acknowledge cycles. When an external interrupt controller is used, or if the integrated interrupt controller is used in Slave mode (see Section 4.4.1) the 80186 will run two interrupt acknowledge cycles back to back.

These cycles are automatically "locked" and will never be separated by bus HOLD. See Section 6.5 on interrupt acknowledge timing for more information concerning interrupt acknowledge timing.

3.3.3 COMING OUT OF HOLD

When the HOLD input goes inactive, the processor lowers its HLDA line in a single clock as shown in Figure 27. If there is pending bus activity, only two T_1 states will be inserted after HLDA goes inactive and status information will go active during the last idle state concerning the bus cycle about to be run (see Sec-

tion 3.1.1). If there are no bus cycles to be run by the CPU, it will continue to float all lines until the last T_1 before it begins its first bus cycle after the HOLD.

A special mechanism exists on the 80C186/80C188 to provide for DRAM refreshing while the bus is in HOLD. If the refresh control unit issues a request to the integrated bus controller while HOLD is in effect, the processor lowers HLDA. It is the responsibility of the external bus master to release the bus by deasserting HOLD so that the refresh cycle can take place (see Figure 28). The external master can then reassume control of the bus subject to the usual requirements placed on the HOLD input.

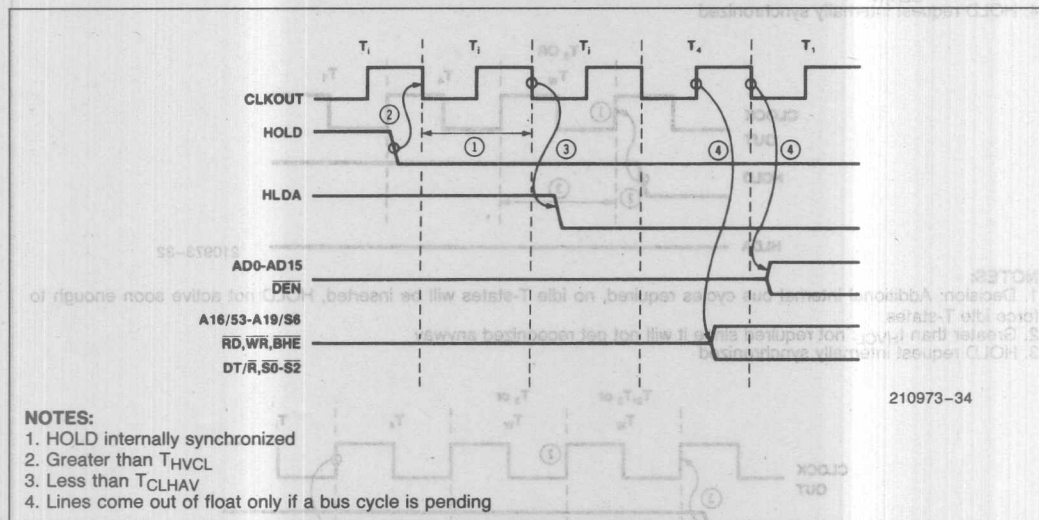


Figure 27. 80186 Coming out of Hold

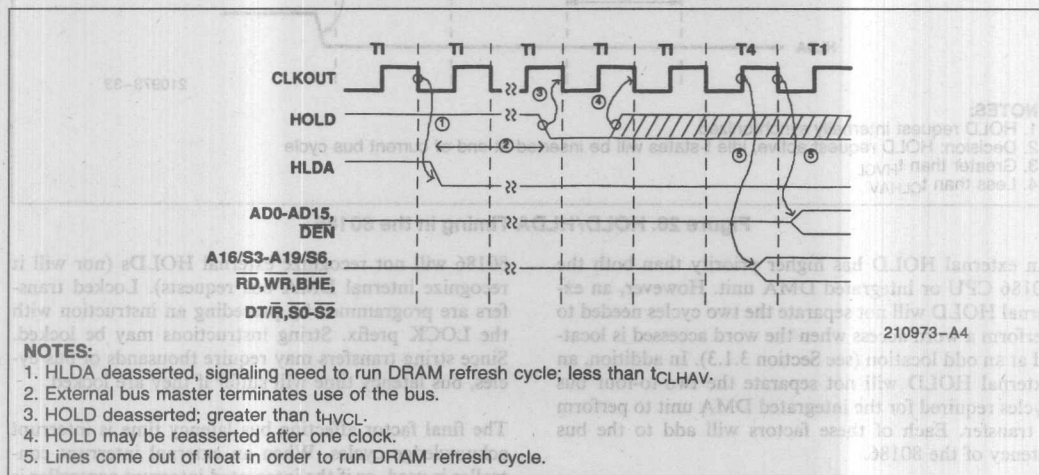


Figure 28. Release of 80C186/80C188 HOLD to Run Refresh Cycle

3.4 Differences between the 8086 Bus and the 80186 Bus

The 80186 bus was defined to be upward compatible with the 8086 bus. As a result, the 8086 bus interface components (the 82C88 bus controller and the 8289 bus arbiter) may be used with the 80186. There are a few significant differences between the two processors which should be considered.

CPU Duty Cycle and Clock Generator

The 80186 employs an integrated clock generator which provides a 50% duty cycle CPU clock. This is different from the 8086, which utilizes an external clock generator to provide 33% ($\frac{1}{3}$ high, $\frac{2}{3}$ low) CPU clock. The following points relate to 80186 clock generation:

- 1) The 80186 uses a crystal or external frequency input twice the desired processor clock frequency.
- 2) No oscillator output is available from the 80186 internal oscillator.
- 3) The 80186 does not provide a clock output at reduced frequency from the 80186. However, a timer output may be easily programmed for this purpose.
- 4) Interfacing the 80186 to devices needing a 33% duty cycle clock (for example, the 8087) is possible, but requires careful timing analysis.
- 5) Care should be exercised not to exceed the drive capability of the 80186 CLKOUT pin.

Local Bus Controller and Control Signals

The 80186 simultaneously provides both local bus controller outputs and status outputs for use with the 82C88 bus controller. This is different from the 8086 where the local bus controller outputs are sacrificed if

status outputs are desired. These differences will manifest themselves in 8086 systems and 80186 systems as follows:

- 1) Because the 80186 can simultaneously provide local bus control signals and status outputs, many systems supporting both a system bus (e.g., a MULTIBUS®) and a local bus will not require two separate external bus controllers, that is, the 80186 bus control signals may be used to control the local bus while the 80186 status signals are concurrently connected to the 82C88 bus controller to drive the control signals of the system bus.
- 2) The ALE signal of the 80186 goes active a clock phase earlier on the 80186 than on the 8086 or 82C88. This minimizes address propagation time through the address latches, since typically the delay time through these latches from inputs valid is less than the propagation delay from the strobe input active.
- 3) The 80186 \overline{RD} input must be tied low to provide queue status outputs from the 80186 (see Figure 29). When so strapped into "queue status mode," the ALE and \overline{WR} outputs provide queue status information. Notice that queue status information is available one clock phase earlier from the 80186 than from the 8086 (see Figure 30).

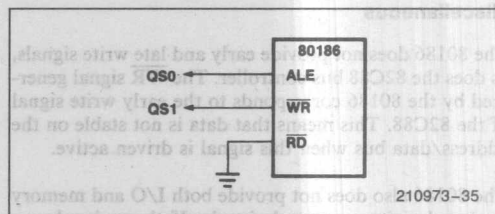
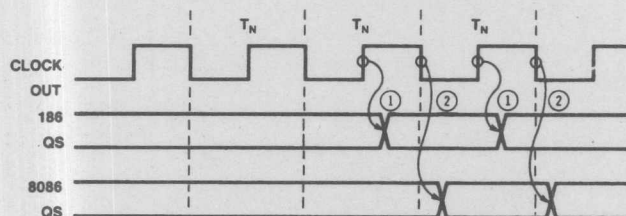


Figure 29. Generating Queue Status Information from the 80186



NOTES:

1. 80186 changes queue status off falling edge of CLK
2. 8086 changes queue status off rising edge of CLK

Figure 30. 80186 and 8086 Queue Status Generation

HOLD/HLDA vs. RQ/GT

As discussed earlier, the 80186 uses a HOLD/HLDA protocol for exchanging bus mastership (like the 8086 in min mode) rather than the RQ/GT protocol used by the 8086 in max mode. This allows compatibility with Intel's bus master peripheral devices (for example the 82586 Ethernet controller or 82730 high performance CRT controller/text coprocessor).

Status Information

The 80186 does not provide S3-S5 status information. On the 8086, S3 and S4 provide information regarding the segment register generating the physical address of the current bus cycle. S5 provides information concerning the state of the interrupt enable flip-flop. These status lines are always low on the 80186.

Status signal S6 indicates whether the current bus cycle is initiated by either the CPU or a DMA device. Subsequently, it is always low on the 8086. On the 80186, it is low whenever the current bus cycle is initiated by the 80186 CPU, and is high when the current bus cycle is initiated by the integrated DMA unit.

Miscellaneous

The 80186 does not provide early and late write signals, as does the 82C88 bus controller. The WR signal generated by the 80186 corresponds to the early write signal of the 82C88. This means that data is not stable on the address/data bus when this signal is driven active.

The 80186 also does not provide both I/O and memory read and write command signals. If these signals are desired, an external 82C88 bus controller may be used, or the S2 signal may be used to synthesize both commands (see Section 3.1.6.1).

4.0 DMA UNIT INTERFACING

The 80186 includes a DMA unit consisting of two independent DMA channels. These channels operate independently of the CPU, and drive all integrated bus interface components (bus controller, chip selects, etc.) exactly as the CPU (see Figure 31). This means that bus cycles initiated by the DMA unit are the same as bus cycles initiated by the CPU (except that S6 = 1 during all DMA initiated cycles). Interfacing the DMA unit itself is very simple, since except for the addition of the DMA request connection, it is exactly the same as interfacing to the CPU.

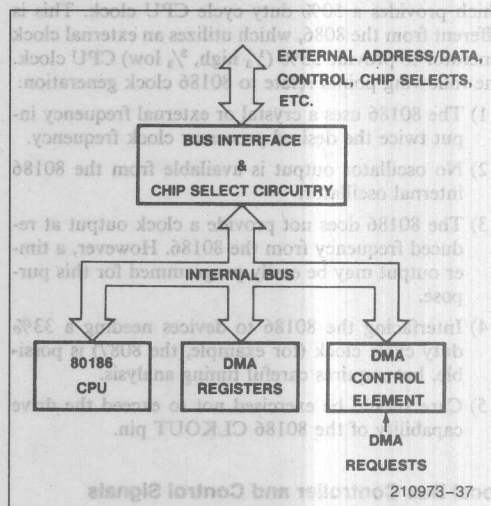


Figure 31. 80186 CPU/DMA Channel Internal Model

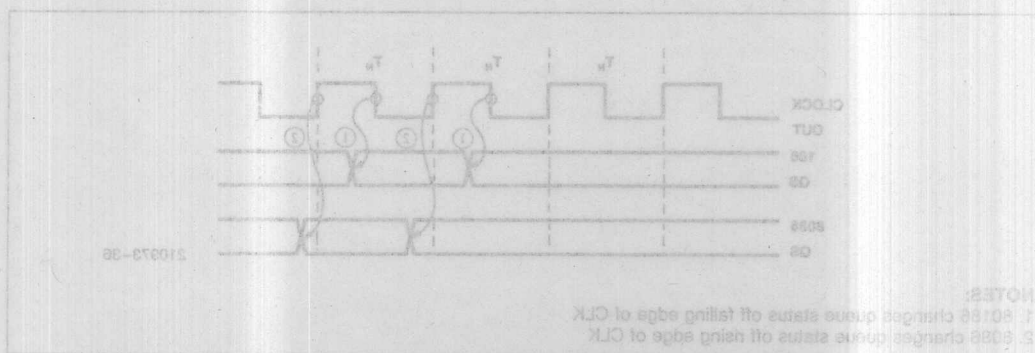


Figure 30. 80186 and 8086 Queue Status Generation

4.1 DMA Features

Each of the two DMA channels provides the following features:

- Independent 20-bit source and destination pointers which access the I/O or memory location from which data will be fetched or to which data will be deposited
- Programmable auto-increment, auto-decrement or neither of the source and destination pointers after each DMA transfer
- Programmable termination of DMA activity after a certain number of DMA transfers
- Programmable CPU interruption at DMA termination
- Byte or word DMA transfers to or from even or odd memory or I/O addresses
- Programmable generation of DMA requests by:
 - 1) the source of the data
 - 2) the destination of the data
 - 3) timer 2 (see Section 5)
 - 4) the DMA unit itself (continuous DMA requests)

4.2 DMA Unit Programming

Each of the two DMA channels contains a number of registers to control channel operation. These registers are included in the 80186 integrated peripheral control block (see Appendix A). These registers include the source and destination pointer registers, the transfer count register and the control register. The layout of the bits in these registers is given in Figures 32 and 33.

The 20-bit source and destination pointers access the complete 1 Mbyte address space of the 80186 and all 20

bits are affected by the auto-increment or auto-decrement unit of the DMA. The address space is seen as a flat, linear array without segments. Even though the usual I/O addressability of the 80186 is 64 Kbytes, it is possible to perform I/O accesses over a 1 Mbyte address range. Therefore, it is important to program the upper four bits of the pointer registers to 0 if routine I/O addresses are desired.

After every DMA transfer the 16-bit DMA transfer count register is decremented by 1, whether a byte transfer or a word transfer has occurred. If the TC bit in the DMA control register is set, the DMA ST/STOP bit (see below) will be cleared when this register goes to 0, causing all DMA activity to cease. A transfer count of zero allows 65536 (2^{16}) transfers.

Upon reset, the contents of the DMA pointer registers and transfer count registers are indeterminate; initialization of all the bits should be practiced.

The DMA control register (see Figure 33) contains bits which control various channel characteristics, including for each of the data source and destination whether the pointer points to memory or I/O space, or whether the pointer will be incremented, decremented or left alone after each DMA transfer. It also contains a bit which selects byte or word transfers. Two synchronization bits determine the source of the DMA requests (see Section 4.7). The TC bit determines whether DMA activity will cease after a programmed number of DMA transfers, and the INT bit enables interrupts to the processor when this has occurred (note that an interrupt will not be generated to the CPU when the transfer count register reaches zero unless both the INT bit and the TC bit are set).

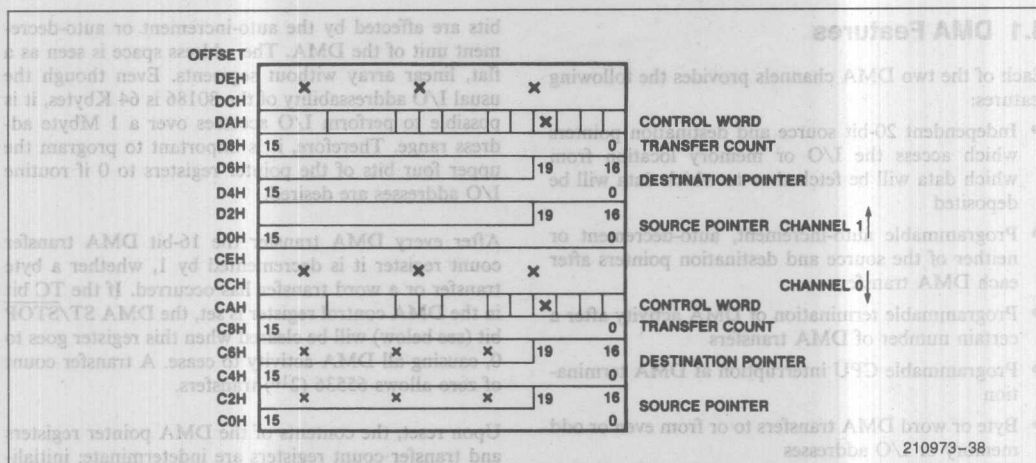


Figure 32. 80186 DMA Register Layout

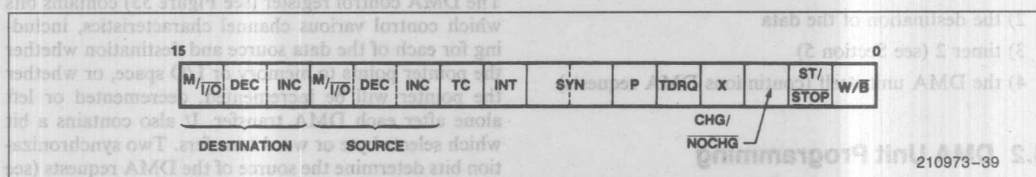


Figure 33. DMA Control Register

The control register also contains a start/stop (ST/STOP) bit which enables DMA transfers. Whenever this bit is set, the channel is armed, that is, a DMA transfer will occur whenever a DMA request is made to the channel. A companion bit, the CHG/NOCHG bit, allows the DMA control register to be changed without modifying the state of the start/stop bit. The ST/STOP bit will only be modified if the CHG/NOCHG bit is also set during the write to the DMA control register. The CHG/NOCHG bit is write only. It will always be read back as a 0. Because DMA transfers could occur immediately after the ST/STOP bit is set, it should only be set after all other DMA controller registers have been programmed. This bit is automatically cleared when the transfer count register reaches zero and the TC bit in the DMA control register is set, or when the transfer count register reaches zero and unsynchronized DMA transfers are programmed.

All DMA unit programming registers are directly accessible by the CPU. This means the CPU can, for example, modify the DMA source pointer register after 137 DMA transfers have occurred, and have the new pointer value used for the 138th DMA transfer. If more than one register in the DMA channel is being modified at any time that a DMA request may be generated and the DMA channel is enabled (the ST/STOP bit in the control register is set), the register programming values should be placed in memory locations and moved into the DMA registers using a locked string move instruction. This will prevent a DMA transfer from occurring after only some of the register values have changed. The above also holds true if a read/modify/write type of operation is being performed (e.g., ANDing off bits in a pointer register in a single AND instruction to a pointer register mapped into memory space).

4.3 DMA Transfers

Every 80186 DMA transfer consists of two independent bus cycles, a fetch cycle and a deposit cycle (see Figure 34). During the fetch cycle, the byte or word data is accessed according to the source pointer register. The data is read into an internal temporary register which is not accessible by the CPU. During the deposit cycle, the data is written to memory or I/O space at the address in the destination pointer register. These two bus cycles cannot be separated by a bus HOLD, a refresh cycle, or any other condition except RESET. DMA bus cycles are identical to bus cycles initiated by the CPU except that the S6 status line is driven to a logic one state.

4.4 DMA Requests

Each DMA channel has a single DMA request line by which an external device may request a DMA transfer. The synchronization bits in the DMA control register determine whether this line is interpreted to be connected to the source or destination of the DMA data. All transfer requests on this line are synchronized internally to the CPU clock before being presented to internal DMA logic. In addition to external requests, DMA requests may be generated whenever the internal Timer 2 times out, or continuously by programming the synchronization bits in the DMA control register for unsynchronized DMA transfers.

4.4.1 DMA REQUEST TIMING AND LATENCY

Before any DMA request can be generated, the 80186 internal bus must be granted to the DMA unit. A certain amount of time is required for the CPU to grant this internal bus to the DMA unit. The time between a DMA request being issued and the DMA transfer being run is known as DMA latency. Many of the issues concerning DMA latency are the same as those concerning bus latency (see Section 3.3.2). The only important difference is that external HOLD and refresh always have bus priority over an internal DMA transfer. Thus, the latency time of an internal DMA cycle will suffer during an external bus HOLD.

Each DMA channel has a programmed priority relative to the other DMA channel. Both channels may be programmed to be the same priority, or one may be programmed to be of higher priority than the other channel. If both channels are active, DMA latency will suffer on the lower priority channel. If both channels are active and both channels are of the same programmed priority, DMA transfer cycles will alternate between the two channels (i.e., the first channel will perform a fetch and deposit, followed by a fetch and deposit by the second channel, etc.).

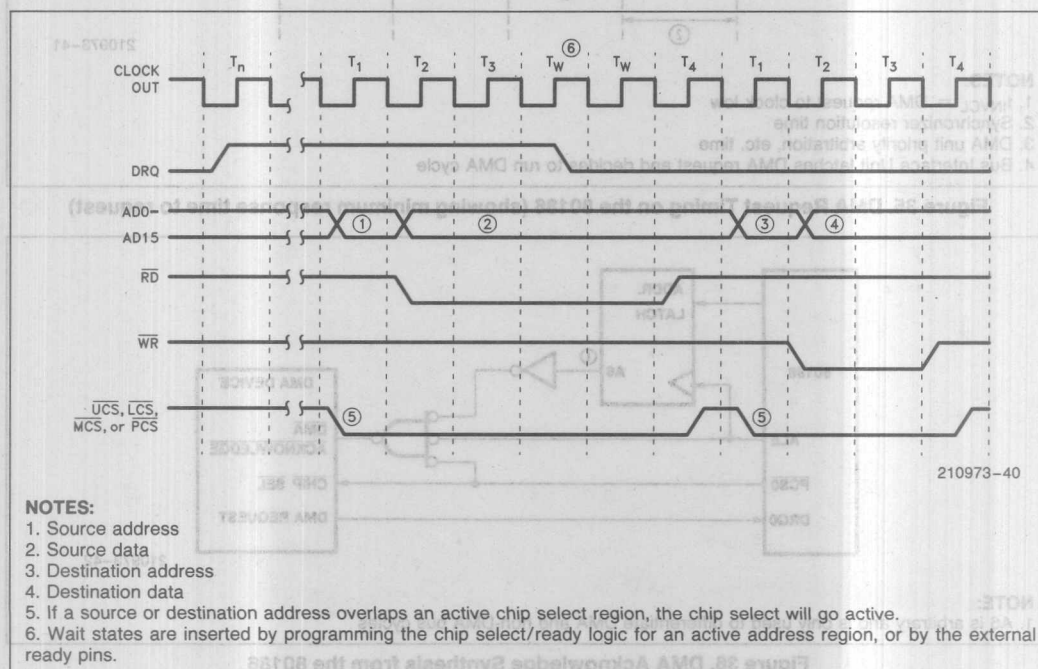


Figure 34. Example DMA Transfer Cycle on the 80186

The minimum timing required to generate a DMA cycle is shown in Figure 35. Note that the minimum time from DRQ becoming active until the beginning of the first DMA cycle is 4 CPU clock cycles. This time is independent of the number of wait states inserted in the bus cycle. The maximum DMA latency is a function of other processor activity.

Also notice that if DRQ is sampled active at 1 in Figure 35, the DMA cycle will be executed, even if the DMA request goes inactive before the beginning of the first DMA cycle. This does not mean that the DMA request is latched into the processor. Quite the contrary, DRQ must be active at a certain time before the end of a bus

cycle for the request to be recognized by the processor. If DRQ goes inactive before that window, then no DMA cycles will be run.

4.5 DMA Acknowledge

The 80186 generates no explicit DMA acknowledge (DACK) signal. Instead, the 80186 performs a read or write directly to the DMA requesting device. If required, a DMA acknowledge signal can be generated by a decode of an address, or by merely using one of the PCS lines (see Figure 36). Note ALE must be used to factor DACK because addresses are not guaranteed stable when chip selects go active.

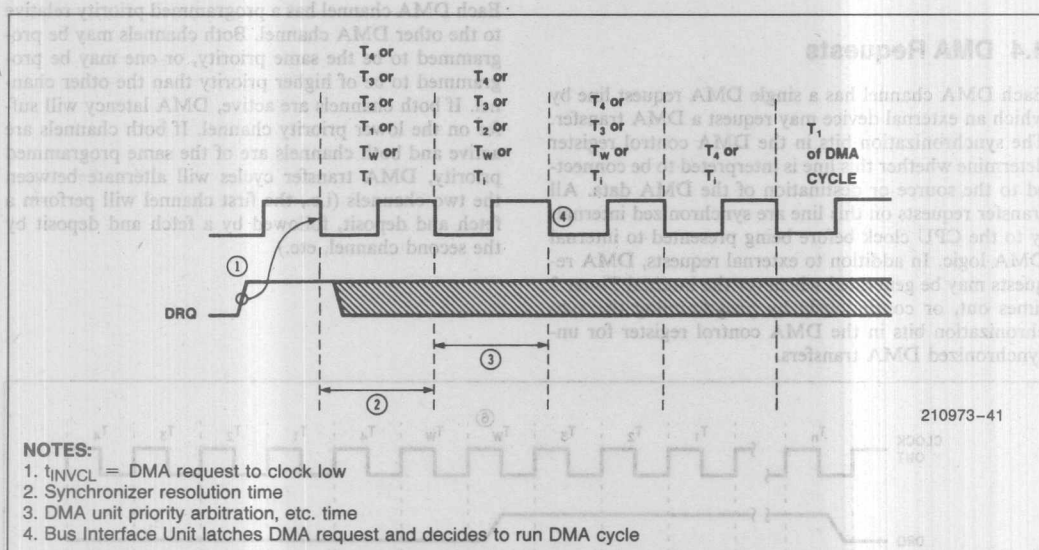


Figure 35. DMA Request Timing on the 80186 (showing minimum response time to request)

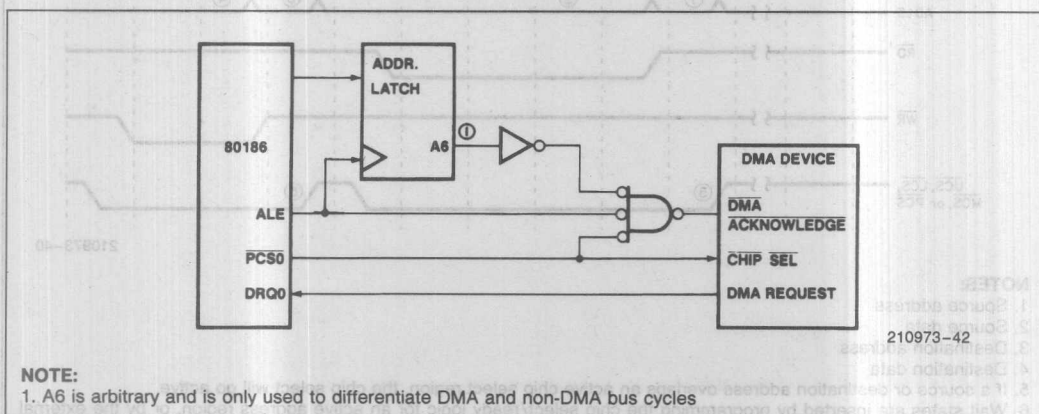


Figure 36. DMA Acknowledge Synthesis from the 80186

4.6 Internally Generated DMA Requests

DMA transfer requests may originate from two of the integrated peripherals in the 80186. The source may be either the DMA control unit or Timer 2.

The DMA channel can be programmed so that whenever Timer 2 reaches its maximum count, a DMA request will be generated. This feature is selected by setting the TDRQ bit in the DMA channel control register. A DMA request generated in this manner will be latched in the DMA controller, so that once the timer request has been generated, it cannot be cleared except by running the DMA cycle or by clearing the TDRQ bits in both DMA control registers. Before any DMA requests are generated in this mode, Timer 2 must be initialized and enabled.

A timer requested DMA cycle being run by either DMA channel will reset the timer request. Thus, if both channels are using it to request a DMA cycle, only one DMA channel will execute a transfer for every timeout of Timer 2. Another implication of having a single bit timer DMA request latch in the DMA controller is that if another Timer 2 timeout occurs before a DMA channel has a chance to run a DMA transfer, the first request will be lost.

The DMA channel can also be programmed to provide its own DMA requests. In this mode, DMA transfer cycles will be run continuously at the maximum bus bandwidth until the preprogrammed number of DMA transfers have occurred. This mode is selected by programming the synchronization bits in the DMA control register for unsynchronized transfers. Note that in this mode, the DMA controller will monopolize the CPU bus, i.e., the CPU will not be able to perform opcode fetching, memory operations, etc., while the DMA transfers are occurring. Also notice that the DMA will only perform the number of transfers indicated in the maximum count register regardless of the state of the TC bit in the DMA control register.

4.7 Externally Synchronized DMA Transfers

There are two types of externally synchronized DMA transfers. These are source and destination synchronized transfers. These modes are selected by programming the synchronization (SYN) bits in the DMA channel control register. The only difference between the two is the time at which the DMA request pin is sampled to determine if another DMA transfer is immediately required after the currently executing DMA transfer. On source synchronized transfers, this is done such that two transfers may occur one immediately after the other, while on destination synchronized transfers a

certain amount of idle time is automatically inserted between two DMA transfers to allow time for the DMA requesting device to drive its DMA request inactive.

4.7.1 SOURCE SYNCHRONIZED DMA TRANSFERS

In a source synchronized DMA transfer, the data source requests the DMA cycle. An example is a floppy disk read from the disk to main memory. In this type of transfer, the device requesting the transfer is read during the fetch cycle of the DMA transfer. Since it takes 4 CPU clock cycles from the time DMA request is sampled to the time the DMA transfer is actually begun, and a bus cycle takes a minimum of 4 clock cycles, the earliest time the DMA request pin will be sampled for another DMA transfer will be at T_1 of the deposit cycle of the DMA transfer (assuming no wait states.) This allows 3 or more CPU clock cycles between the time the DMA requesting device receives an acknowledge to its DMA request (around the beginning of T_2 of the DMA fetch cycle), and the time it must drive this request inactive (assuming no wait states) to insure that another DMA transfer is not performed if it is not desired (see Figure 37).

4.7.2 DESTINATION SYNCHRONIZED DMA TRANSFERS

In destination synchronized DMA transfers, the data destination requests the DMA transfer. An example of this would be a floppy disk write from main memory to the disk. In this type of transfer, the device requesting the transfer is written during the deposit cycle of the DMA transfer. This causes a problem, since the DMA requesting device will not receive notification of the DMA cycle being run until 3 clock cycles before the end of the DMA transfer (if no wait states are being inserted into the deposit cycle of the DMA transfer) and it takes 4 clock cycles to determine if another DMA cycle should run immediately following the current transfer. To get around this problem, the DMA unit relinquishes the bus after each destination synchronized DMA transfer for at least 2 CPU clock cycles to allow the requesting device time to drop its DMA request if it does not immediately desire another DMA transfer. When the bus is relinquished by the DMA unit, the CPU may resume bus operation. Typically, a CPU initiated bus cycle is inserted between each destination synchronized DMA transfer. If no CPU bus activity is required, however, the DMA unit inserts only 2 CPU clock cycles between the deposit cycle of one DMA transfer and the fetch cycle of the next DMA transfer. This means that the requesting device must drop its request line at least two clock cycles before the end of the deposit cycle regardless of the number of wait states inserted. Figure 37 shows the DMA request going away too late to prevent the immediate generation of another DMA transfer. Any wait states inserted

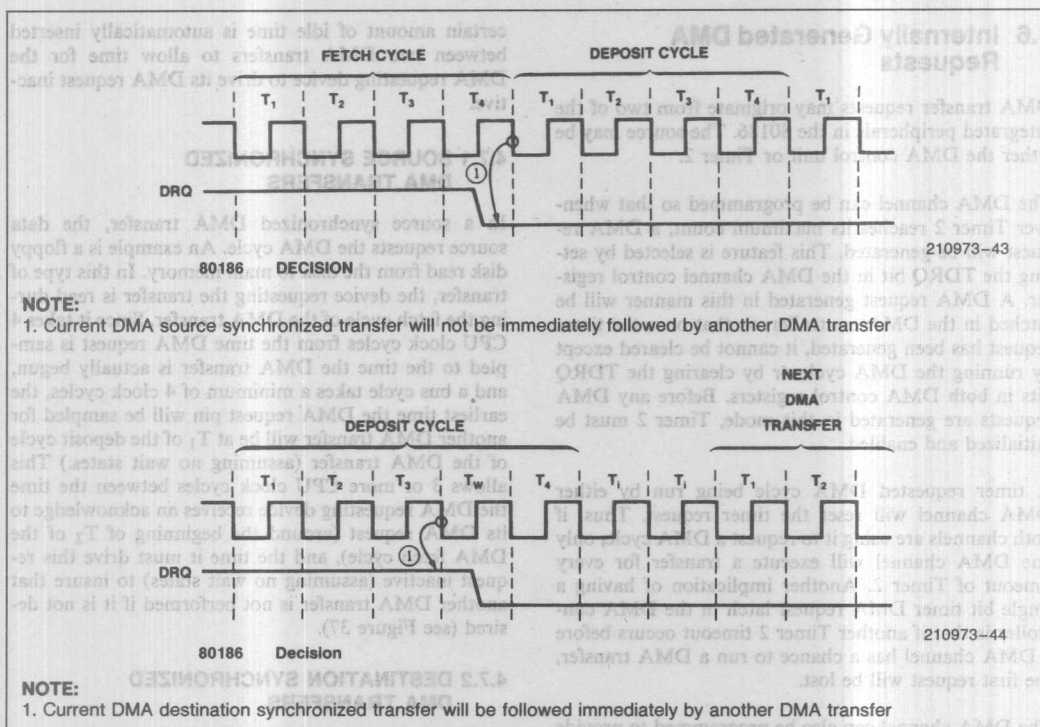


Figure 37. Source & Destination Synchronized DMA Request Timing

in the deposit cycle of the transfer lengthen the amount of time from the beginning of the deposit cycle to the time DRQ is sampled for another DMA transfer. Thus, if the amount of time a device requires to drop its request after receiving an acknowledge from the 80186 is longer than the 0 wait state 80186 maximum (about 1 clock), wait states can be inserted into the DMA cycle to lengthen the amount of time the device has to drop its request after receiving the DMA acknowledge.

4.8 DMA Halt and NMI

Whenever a Non-Maskable Interrupt is received by the 80186, all DMA activity will be suspended at the end of the current DMA transfer. This is performed by the NMI automatically setting the DMA Halt (DHLT) bit in the interrupt controller status register (see Section 6.3.7). The timing of NMI required to prevent a DMA cycle from occurring is shown in Figure 38. After the NMI has been serviced, the DHLT bit can be cleared by the programmer to resume DMA activity (i.e., it is not automatically cleared when entering the NMI service routine). The DHLT bit is automatically cleared when the IRET instruction is executed. In either case, DMA activity resumes exactly as it left off, i.e., none of the DMA control registers are modified. This DHLT bit may also be set by the programmer to prevent

DMA activity during critical sections of code. The DHLT bit does not function when the integrated interrupt controller is configured for Slave Mode.

4.9 Example DMA Interfaces

4.9.1 8272 FLOPPY DISK INTERFACE

An example DMA interface to the 8272 Floppy Disk Controller is shown in Figure 39. This shows how a typical DMA device can be interfaced to the 80186. An example floppy disk software driver for this interface is given in Appendix C.

The data lines of the 8272 are connected, through buffers, to the 80186 AD0-AD7 lines. The buffers are required because the 8272 will not float its output drivers quickly enough to prevent contention with the 80186 upon the next bus cycle (see Section 3.1.5).

DMA acknowledge for the 8272 is driven by an address decode within the region assigned to PCS2. If PCS2 is assigned to be active between I/O locations 0500H and 057FH, then an access to I/O location 0500H will enable only the chip select, while an access to I/O location 0501H will enable both the chip select and the DMA acknowledge. Remember, ALE must be factored

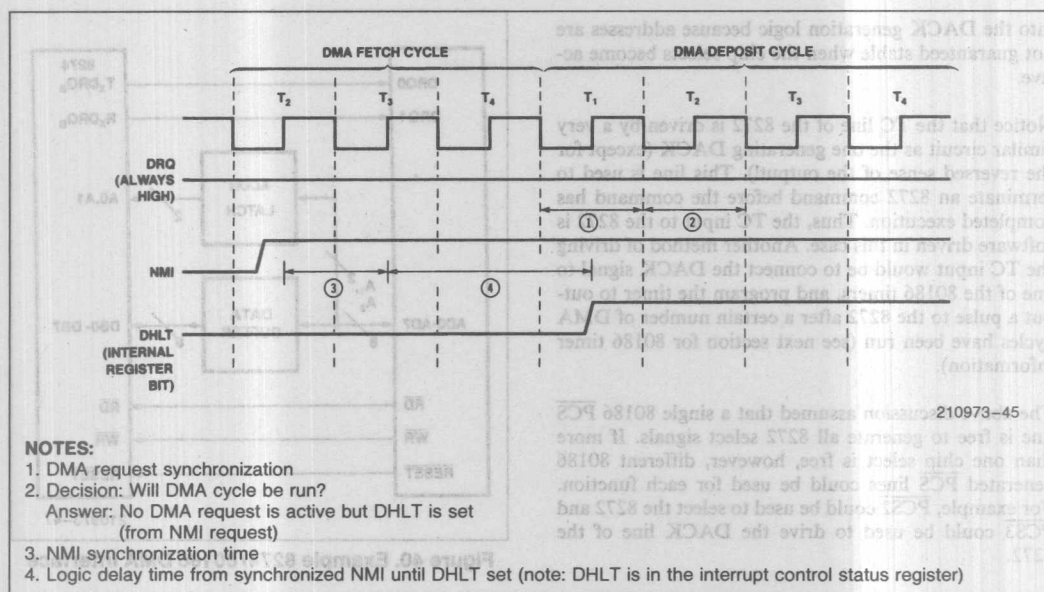


Figure 38. NMI and DMA Interaction

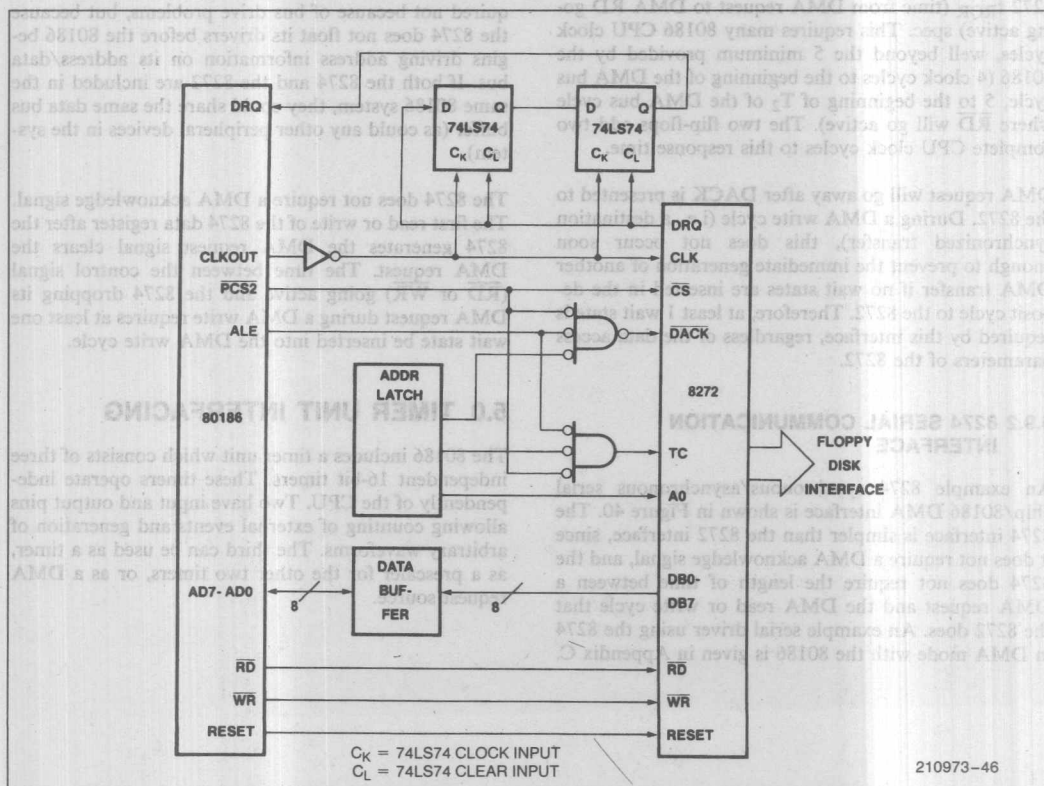


Figure 39. Example 8272/80186 DMA Interface

5.1 Timer Operation

The internal timer unit on the 80186 can be modeled by a single counter element, time multiplexed to three register banks, each of which contains different control and count values. These register banks are, in turn, dual ported between the counter element and the 80186 CPU (see Figure 41). Figure 42 shows the timer element sequencing, and the subsequent constraints on input and output signals. There is no connection between the sequencing of the counter element through the timer register banks and the Bus Interface Unit's sequencing through T-states. Timer operation and bus interface operation are completely asynchronous.

5.2 Timer Unit Programming

Each timer is controlled by a block of registers (see Figure 43). Each of these registers can be read or written whether or not the timer is operating. All processor accesses to these registers are synchronized to all counter element accesses to these registers, meaning that one will never read a count register in which only half of the bits have been modified. Because of this synchronization, one wait state is automatically inserted into any access to the timer registers. Unlike the DMA unit, locking accesses to timer registers will not prevent the timer's counter elements from accessing the timer registers.

Each timer has a 16-bit count register which is incremented for each timer event. A timer event can be

a low-to-high transition on a TIMERIN pin (for Timers 0 and 1), a pulse generated every fourth CPU clock, or a time out of Timer 2 (for Timers 0 and 1). Because the count register is 16 bits wide, up to 65536 (2^{16}) timer events can be counted. Upon RESET, the contents of the count registers are indeterminate and they should be initialized to zero before any timer operation.

Each timer includes a maximum count register. Whenever the timer count register is equal to the maximum count register, the count register resets to zero, so the maximum count value can never be stored in the count register. This maximum count value may be written while the timer is operating. A maximum count value of 0 implies a maximum count of 65536, a maximum count value of 1 implies a maximum count of 1, etc. Only equivalence between the count value and the maximum count register value is checked. This means that the count value will not be cleared if the value in the count register is greater than the value in the maximum count register. This situation only occurs by programmer intervention, either by setting the value in the count register greater than the value in the maximum count register, or by setting the value in the maximum count register to be less than the value in the count register. If the timer is programmed in this way, it will count to the maximum possible count (FFFFH), increment to 0, then count up to the value in the maximum count register. The TC bit in the timer control register will not be set when the counter overflows to 0, nor will an interrupt be generated from the timer unit.

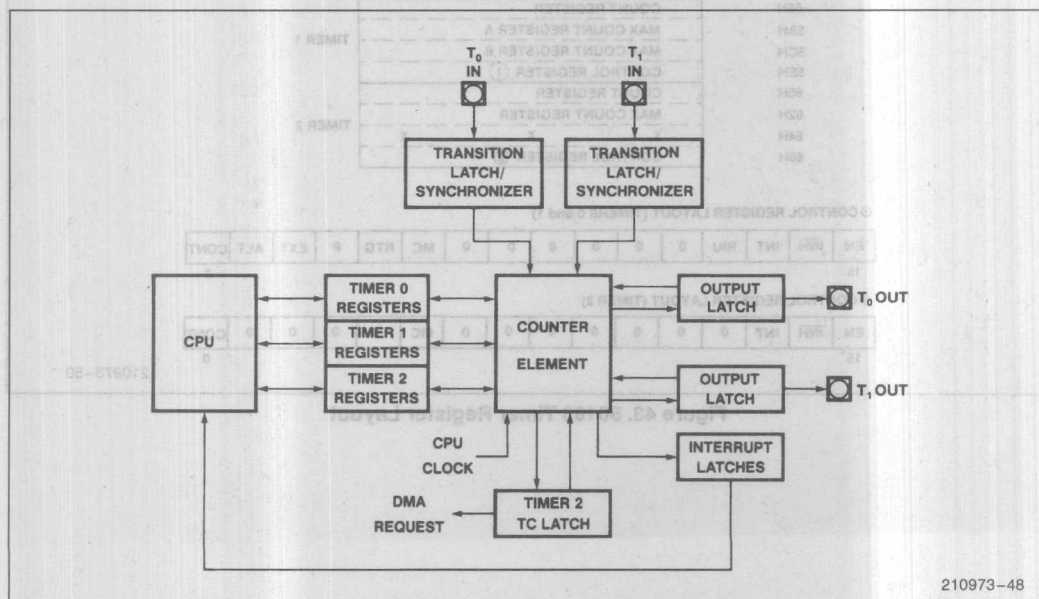


Figure 41. 80186 Timer Model

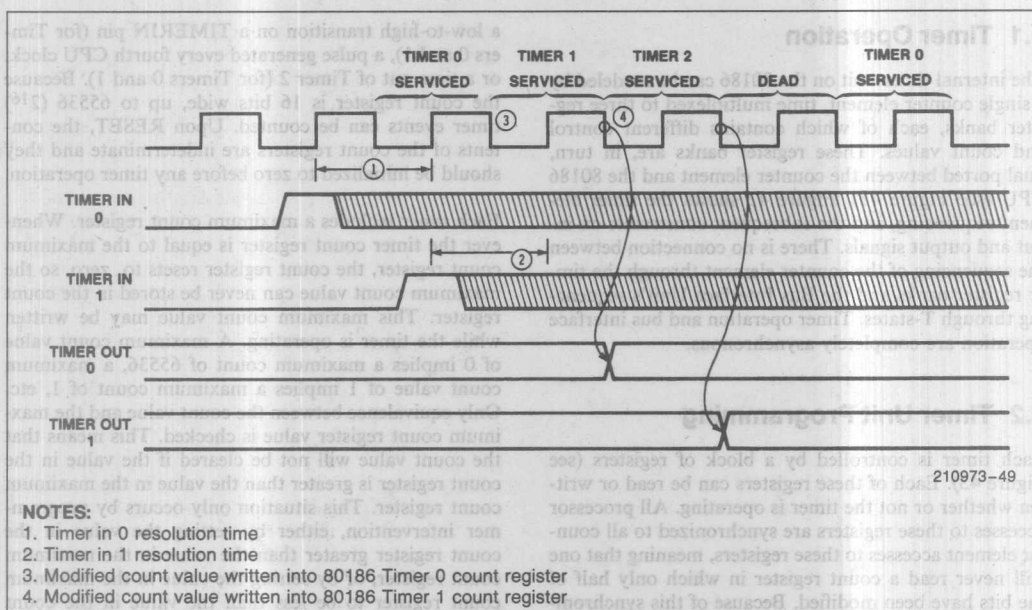


Figure 42. 80186 Counter Element Multiplexing and Timer Input Synchronization

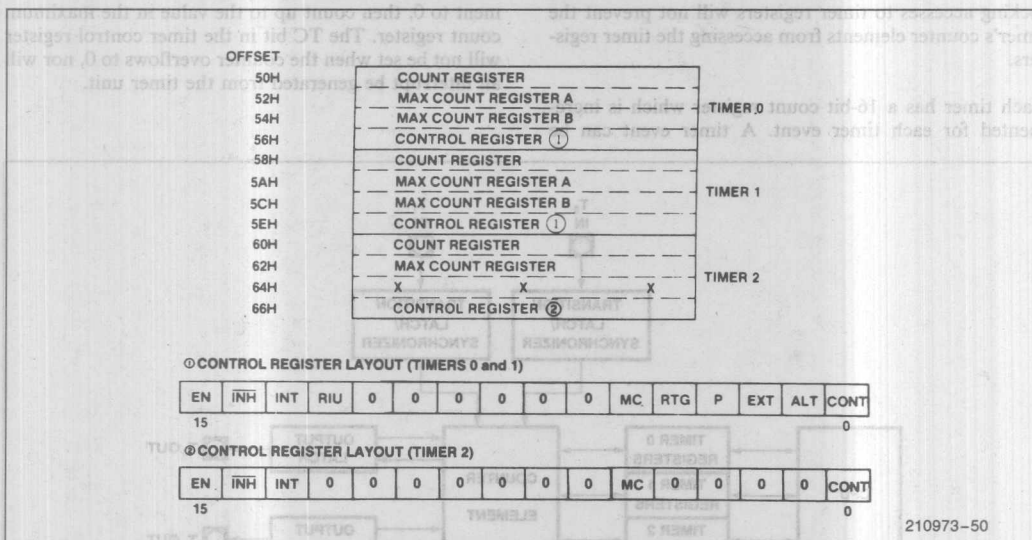


Figure 43. 80186 Timer Register Layout

Timers 0 and 1 each contain an additional maximum count register. When both maximum count registers are used, the timer will first count up to the value in maximum count register A, reset to zero, count up to the value in maximum count register B, and reset to zero again. The ALTErnatE bit in the timer control register determines whether one or both maximum count registers are used. If this bit is low, only maximum count register A is used; maximum count register B is ignored. If it is high, both registers are used. The RIU (register in use) bit in the timer control register indicates which maximum count register is currently being used. This bit is 0 when maximum count register A is being used, 1 when maximum count register B is being used. The RIU bit is read only. It will always be read 0 in single maximum count register mode (since only maximum count register A will be used).

Each timer can generate an interrupt whenever the timer count value reaches a maximum count value. Interrupts result whenever the timer count matches maximum count A (for Timer 2 or Timers 0 and 1 in single max count mode) and whenever the timer count matches maximum count B (for Timers 0 and 1 in dual max count mode). In addition, the MC (maximum count) bit in the timer control register is set whenever the timer count reaches a maximum count value. This bit is never automatically cleared, i.e., programmer intervention is required. If a timer generates a second interrupt request before the first interrupt request has been serviced, the first interrupt request to the CPU will be lost.

Each timer has an ENable bit in the timer control register. The timer will count timer events only when this bit is set. Any write to the timer control register will modify the ENable bit only if the INHibit bit is also set. The INHibit bit in the timer control register allows selective updating of the timer ENable bit. The value of the INHibit bit is not stored in a write to the timer control register; it will always be read as a logic zero.

Each timer has a CONTInuous bit in the timer control register. If this bit is cleared, the timer ENable bit will be automatically cleared at the end of each timing cycle. If a single maximum count register is used, the end of a timing cycle occurs when the count value resets to zero after reaching the value in maximum count register A. If dual maximum count registers are used, the end of a timing cycle occurs when the count value resets to zero after reaching the value in maximum count register B. If the CONTInuous bit is set, the ENable bit will never be automatically reset. Thus, after each timing cycle, another timing cycle will automatically begin. For example, in single maximum count register mode, the timer will count up to the value in maximum count register A, reset to zero, ad infinitum. In dual maximum count register mode, the timer will count up to the value in maximum count register A, reset to zero,

count up to the value in maximum count register B, reset to zero, count up to the value in maximum count register A, reset to zero, et cetera.

5.3 Timer Events

Each timer counts events. All timers can use a transition of the CPU clock as an event. Because of the counter element multiplexing, the timer count value will be incremented every fourth CPU clock. For Timer 2, this is the only timer event which can be used. For Timers 0 and 1, this event is selected by clearing the EXTErnal and Prescaler bits in the timer control register.

Timers 0 and 1 can use Timer 2 reaching its maximum count as a timer event. This is selected by clearing the EXTErnal bit and setting the Prescaler bit in the timer control register. When this is done, the timer will increment whenever Timer 2 resets to zero having reached its own maximum count. Note that Timer 2 must be initialized and running in order to increment the value in the other timer/counter.

Timers 0 and 1 can also be programmed to count low-to-high transitions on the external input pin. Each transition on the external pin is synchronized to the 80186 clock before it is presented to the timer circuitry, (see Appendix B for information on 80186 synchronizers). The timer counts transitions on the input pin: the input value must go low, then go high to cause the timer increment. Transitions on this line are latched. Because of the counter element multiplexing, the maximum rate at which the timer can count is 1/4 of the CPU clock rate.

5.4 Timer Input Pin Operation

Timers 0 and 1 each have individual timer input pins. All low-to-high transitions on these input pins are synchronized, latched, and presented to the counter element when the particular timer is being serviced by the counter element.

Signals on this input can affect timer operation in three different ways. The manner in which the pin signals are used is determined by the EXTErnal and RTG (retrigger) bits in the timer control register. If the EXTErnal bit is set, transitions on the input pin will cause the timer count value to increment if the timer is enabled (the ENable bit in the timer control register is set). Thus, the timer counts external events. If the EXTErnal bit is cleared, all timer increments are caused by either the CPU clock or by Timer 2 timing out. In this mode, the RTG bit determines whether the input pin will enable timer operation, or whether it will retrigger timer operation.

low, the timer will count internal timer events only when the timer input pin is high and the ENable bit in the timer control register is set. Note that in this mode, the pin is level sensitive, not edge sensitive. A low-to-high transition on the timer input pin is not required to enable timer operation. If the input is tied high, the timer will be continually enabled. The timer enable input signal is completely independent of the ENable bit in the timer control register: both must be high for the timer to count. Example uses for the timer in this mode would be a real time clock or a baud rate generator.

When the EXTERNAL bit is low and the RTG bit is high, every low-to-high transition on the timer input pin causes the timer count register to reset to zero. This mode of operation can be used to generate a retriggerable digital one-shot. After the timer is enabled (i.e., the ENable bit in the timer control register is set), timer operation (counting) will begin only after the first low-to-high transition of the timer input pin has been detected. If another low-to-high transition occurs on the input pin before the end of the timer cycle, the timer will reset to zero and begin the timer cycle again. A timer cycle is defined as the time the timer is counting from 0 to the maximum count (either max count A or max count B). This means that in the dual max count mode, the RIU bit is not set if the timer is reset by the low-to-high transition on the input pin. Should a timer reset occur when RIU is set (indicating max count B), the timer will again begin to count up to max count B before resetting the RIU bit. Thus, when the ALTer-nate bit is set, a timer reset will retrigger (or extend) the duration of the current max count in use (which means that either the low or high level of the timer output will be extended). If the CONTInuous bit in the timer control register is cleared, the timer ENable bit will automatically be cleared whenever a timer cycle has been completed (max count is reached). If the CONTInuous

reset to zero and begin another timer cycle whenever the current cycle has completed.

5.5 Timer Output Pin Operation

Timers 0 and 1 each have a timer output pin which can perform two functions at programmer option. The first is a single pulse indicating the end of a timing cycle. The second is a level indicating the maximum count register currently being used. The timer outputs operate as outlined below whether internal or external clocking of the timer is used. If external clocking is used, however, the user should remember that the time between an external transition on the timer input pin and the time this transition is reflected in the timer out pin will vary depending on when the input transition occurs relative to the timer being serviced by the counter element.

When the timer is in single maximum count register mode, the timer output pin will go low for a single CPU clock one clock after the timer is serviced by the counter element where maximum count is reached (see Figure 44). This mode is useful when using the timer as a baud rate generator.

When the timer is programmed in dual maximum count register mode, the timer output pin indicates which maximum count register is being used. It is low if maximum count register B is being used and high if maximum count register A is being used. If the timer is programmed in continuous mode (the CONTInuous bit in the timer control register is set), this pin could generate a waveform of almost any duty cycle. For example, if maximum count register A contained 10 and maximum count register B contained 20, a 33% duty cycle waveform would be generated.

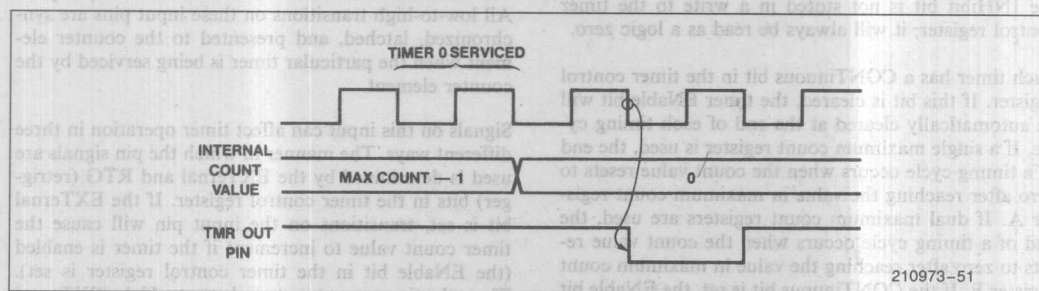


Figure 44. 80186 Timer Out Signal

5.6 Sample 80186 Timer Applications

The 80186 timers can be substituted in almost any application for a discrete timer circuit. Such applications include baud rate generation, digital one-shots, pulse width modulation, event counters and pulse width measurement.

5.6.1 80186 TIMER REAL TIME CLOCK

The sample program in appendix D shows the 80186 timer being used with the 80186 CPU to form a real time clock. In this implementation, Timer 2 is programmed to provide an interrupt to the CPU every millisecond. The CPU then increments memory based clock variables.

5.6.2 80186 TIMER BAUD RATE GENERATOR

The 80186 timers can also be used as baud rate generators for serial communication controllers (e.g., the 8274). Figure 46 shows this simple connection, and the code to program the timer as a baud rate generator is included in Appendix D.

5.6.3 80186 TIMER EVENT COUNTER

The 80186 timer can be used to count events. Figure 47 shows a hypothetical set up in which the 80186 timer will count the interruptions in a light source. The number of interruptions can be read directly from the count register of the timer, since the timer counts up, i.e., each interruption in the light source will cause the timer count value to increase. The code to set up the 80186 timer in this mode is included in Appendix D.

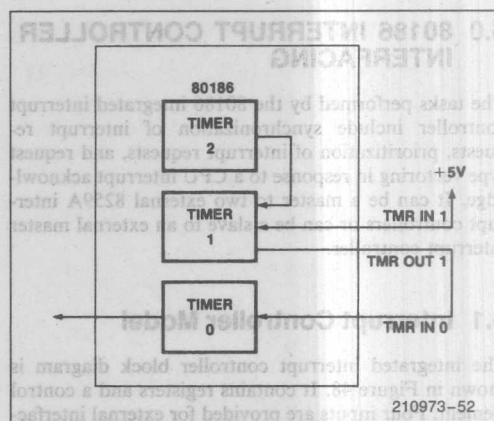


Figure 45. 80186 Real Time Clock

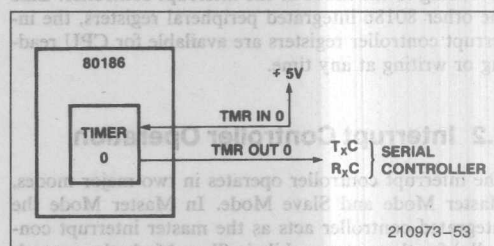


Figure 46. 80186 Baud Rate Generator

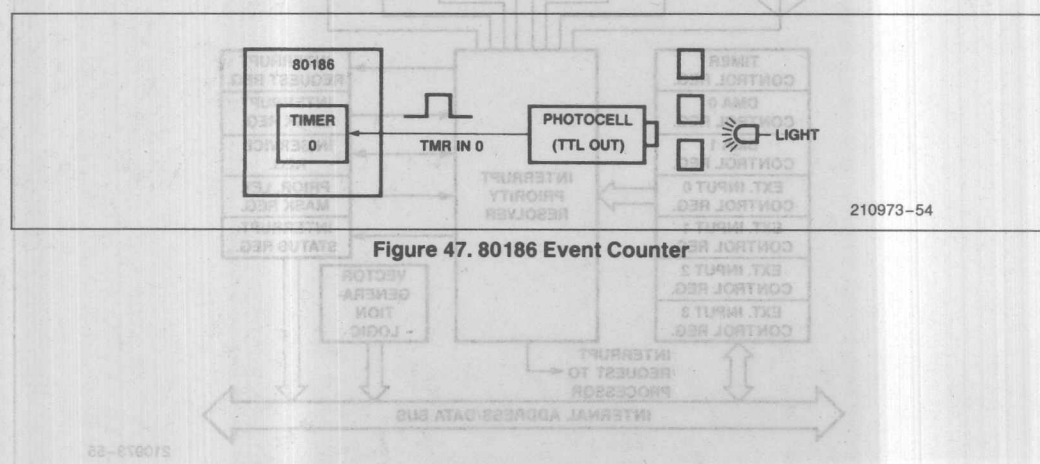


Figure 47. 80186 Event Counter

6.0 80186 INTERRUPT CONTROLLER INTERFACING

The tasks performed by the 80186 integrated interrupt controller include synchronization of interrupt requests, prioritization of interrupt requests, and request type vectoring in response to a CPU interrupt acknowledge. It can be a master to two external 8259A interrupt controllers or can be a slave to an external master interrupt controller.

6.1 Interrupt Controller Model

The integrated interrupt controller block diagram is shown in Figure 48. It contains registers and a control element. Four inputs are provided for external interfacing to the interrupt controller. Their functions change according to the mode of the interrupt controller. Like the other 80186 integrated peripheral registers, the interrupt controller registers are available for CPU reading or writing at any time.

6.2 Interrupt Controller Operation

The interrupt controller operates in two major modes, Master Mode and Slave Mode. In Master Mode the integrated controller acts as the master interrupt controller for the system, while in Slave Mode the controller operates as a slave to an external master inter-

rupt controller. Some of the interrupt controller registers and interrupt controller pins change definition between these two modes. The difference is when in Master Mode, the interrupt controller presents its interrupt input directly to the 80186 CPU, while in Slave Mode the interrupt controller presents an interrupt output to an external controller (which then presents its interrupt input to the 80186 CPU). Placing the interrupt controller in Slave Mode is done by setting the SLAVE/MASTER bit in the peripheral control block pointer (see Appendix A).

6.3 Interrupt Controller Unit Programming

The interrupt controller has a number of registers which control its operation (see Figure 49). Some of these change their function between the two major modes of the interrupt controller. The differences are indicated in the following section. If not indicated, the function and implementation of the registers is the same in the two modes of operation. The interaction among the various interrupt controller registers is shown in the flowcharts in Figures 57 and 58.

6.3.1 CONTROL REGISTERS

Each source of interrupt to the 80186 has a control register in the internal controller. These registers

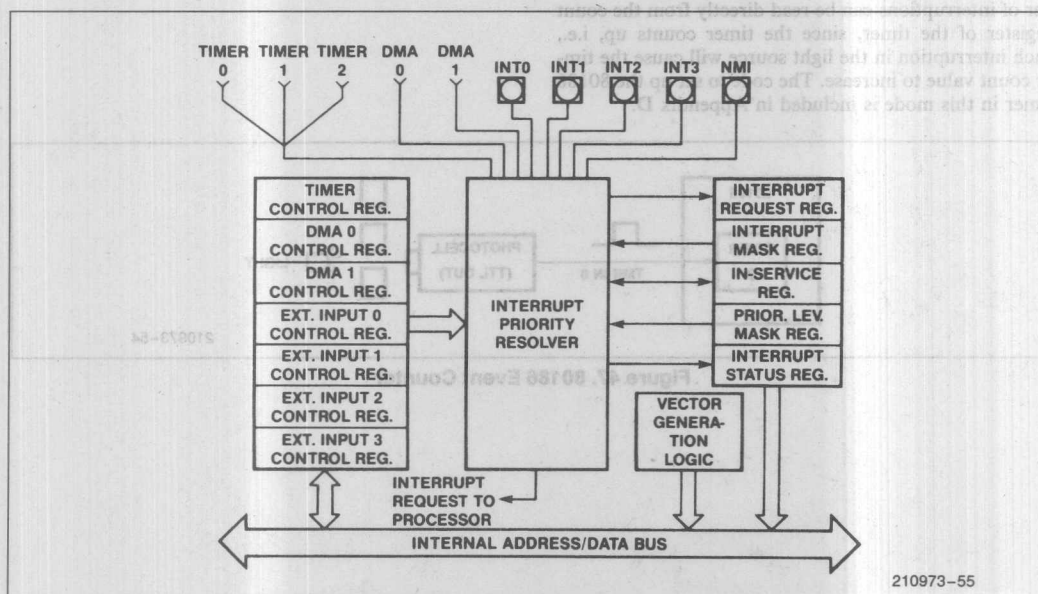


Figure 48. 80186 Interrupt Controller Block Diagram

MASTER MODE	OFFSET ADDRESS	SLAVE MODE
INT3 CONTROL REGISTER	3EH	①
INT2 CONTROL REGISTER	3CH	①
INT1 CONTROL REGISTER	3AH	TIMER 2 CONTROL REGISTER
INT0 CONTROL REGISTER	38H	TIMER 1 CONTROL REGISTER
DMA1 CONTROL REGISTER	36H	DMA1 CONTROL REGISTER
DMA0 CONTROL REGISTER	34H	DMA0 CONTROL REGISTER
TIMER CONTROL REGISTER	32H	TIMER 0 CONTROL REGISTER
INTERRUPT CONTROLLER STATUS REGISTER	30H	INTERRUPT CONTROLLER STATUS REGISTER
INTERRUPT REQUEST REGISTER	2EH	INTERRUPT REQUEST REGISTER
IN-SERVICE REGISTER	2CH	IN SERVICE REGISTER
PRIORITY MASK REGISTER	2AH	PRIORITY MASK REGISTER
MASK REGISTER	28H	MASK REGISTER
POLL STATUS REGISTER	26H	①
POLL REGISTER	24H	①
EOI REGISTER	22H	SPECIFIC EOI REGISTER
①	20H	INTERRUPT VECTOR REGISTER

210973-56

NOTE:

1. Unsupported in this mode: values written may or may not be stored

Figure 49. 80186 Interrupt Controller Registers

contain three bits which select one of eight interrupt priority levels for the device (0 is highest priority, 7 is lowest priority), and a mask bit to enable the interrupt (see Figure 50). When the mask bit is zero, the interrupt is enabled, when it is one, the interrupt is masked.

There are seven control registers in the 80186 integrated interrupt controller. In Master Mode, four of these serve the external interrupt inputs, one each for the two DMA channels, and one for the collective timer interrupts. In Slave Mode, the external interrupt inputs are not used, so each timer has its own individual control register.

6.3.2 REQUEST REGISTER

The interrupt controller includes an interrupt request register (see Figure 51). This register contains seven active bits, one for every interrupt source with an interrupt control register. Whenever an interrupt request is made, the bit in the interrupt request register is set regardless of whether the interrupt is enabled. These interrupt request bits are automatically cleared when the

interrupt is acknowledged. The D1 and D0 bits of the request register can also be set (requesting a DMA interrupt), or cleared (removing a DMA interrupt request) by programming.

6.3.3 MASK REGISTER AND PRIORITY MASK REGISTER

The interrupt controller mask register (see Figure 51) contains a mask bit for each interrupt source associated with an interrupt control register. The bit for an interrupt source in the mask register is the same bit as provided in the interrupt control register; modifying a mask bit in the control register will also modify it in the mask register, and vice versa.

The interrupt priority mask register (see Figure 52) contains three bits which indicate the lowest priority an interrupt may have that will cause an interrupt request to actually be serviced. Interrupts received which have a lower priority will be masked. Upon reset this register is set to the lowest priority of 7 to enable interrupts of any priority. This register may be read or written.

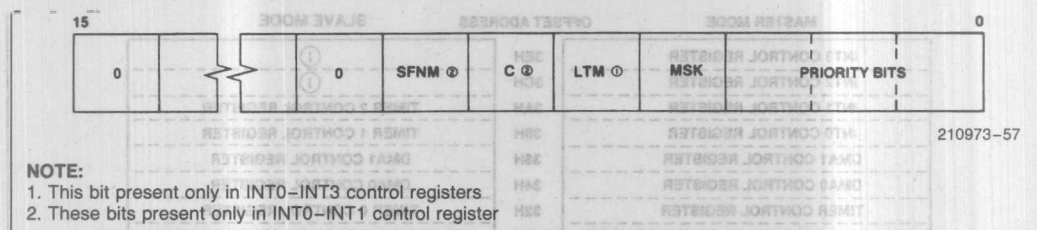


Figure 50. Interrupt Controller Control Register

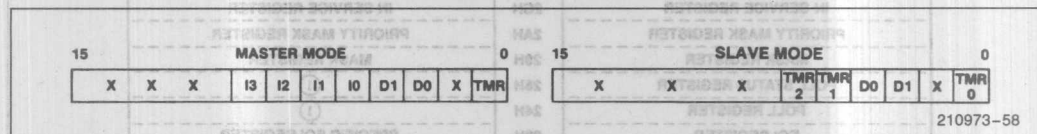


Figure 51. 80186 Interrupt Controller In-Service, Interrupt Request and Mask Register Format

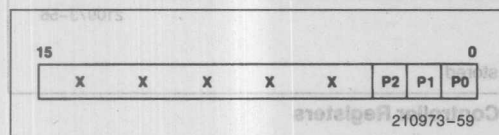


Figure 52. 80186 Interrupt Controller Priority Mask Register Format

6.3.4 IN-SERVICE REGISTER

The interrupt controller contains an in-service register (see Figure 51). A bit in the in-service register is associated with each interrupt control register so that when an interrupt request by the device associated with the control register is acknowledged by the processor (either by the processor running the interrupt acknowledge or by the processor reading the interrupt poll register) the bit is set. The bit is reset when the CPU issues an End Of Interrupt to the interrupt controller. This register may be both read and written, i.e., the CPU may set in-service bits without an interrupt ever occurring, or may reset them without using the EOI function of the interrupt controller.

6.3.5 POLL AND POLL STATUS REGISTERS

The interrupt controller contains both a poll register and a poll status register (see Figure 53). These re-

gisters contain the same information. They have a single bit to indicate an interrupt is pending. This bit is set if an interrupt of sufficient priority has been received. It is automatically cleared when the interrupt is acknowledged. If an interrupt is pending, the remaining bits contain information about the highest priority pending interrupt. These registers are read-only.

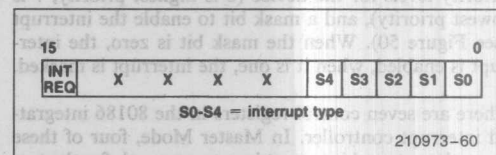


Figure 53. 80186 Poll & Poll Status Register Format

Reading the poll register will acknowledge the pending interrupt to the interrupt controller just as if the processor had acknowledged the interrupt through interrupt acknowledge cycles. The processor will not actually run any interrupt acknowledge cycles, and will not vector through a location in the interrupt vector table. The contents of the interrupt request, in-service, poll, and poll status registers will change appropriately. Reading the poll status register will merely transmit the status of the polling bits without modifying any of the other interrupt controller registers.

The poll and poll status registers are not supported in Slave Mode. The state of the bits in these registers in Slave Mode is not defined.

6.3.6 END OF INTERRUPT REGISTER

The interrupt controller contains an End Of Interrupt register (see Figure 54). The programmer issues an End Of Interrupt (EOI) to the controller by writing to this register. After receiving the EOI, the interrupt controller automatically resets the in-service bit for the interrupt. The value of the word written to this register determines whether the EOI is specific or non-specific. A non-specific EOI is specified by setting the non-specific bit in the word written to the EOI register. In a non-specific EOI, the in-service bit of the highest priority interrupt set is automatically cleared, while a specific EOI allows the in-service bit cleared to be explicitly specified. If the highest priority interrupt is reset, the poll and poll status registers change to reflect the next lowest priority interrupt to be serviced. If a less than highest priority interrupt in-service bit is reset, the priority poll and poll status registers will not be modified (because the highest priority interrupt to be serviced has not changed). Only the specific EOI is supported in Slave Mode. This register is write only.

6.3.7 INTERRUPT STATUS REGISTER

The interrupt controller also contains an interrupt status register (see Figure 55). This register contains

four programmable bits. Three bits show which timer is causing an interrupt. This is required because in master mode, the timers share a single interrupt control register. A bit in this register is set to indicate which timer has generated an interrupt. The bit associated with a timer is automatically cleared after the interrupt request for the timer is acknowledged. More than one of these bits may be set at a time. The fourth bit is the DMA halt bit (not implemented in Slave Mode). When set, this bit prevents any DMA activity. It is automatically set whenever a NMI is received by the interrupt controller. It can also be set by the programmer. This bit is automatically cleared whenever the IRET instruction is executed. All implemented bits in the interrupt status register are read/write. Do not perform the write operation when interrupts from the timer/counters are possible; a conflict with internal use of the register may lead to incorrect timer interrupt processing.

6.3.8 INTERRUPT VECTOR REGISTER

In Slave Mode only, the interrupt controller contains an interrupt vector register (see Figure 56). This register specifies the 5 most significant bits of the interrupt type vector placed on the CPU bus in response to an interrupt acknowledgement (the lower 3 significant bits of the interrupt type are determined by the priority level of the device causing the interrupt in Slave Mode).

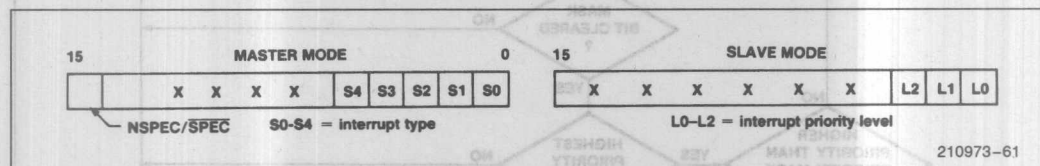


Figure 54. 80186 End of Interrupt Register Format

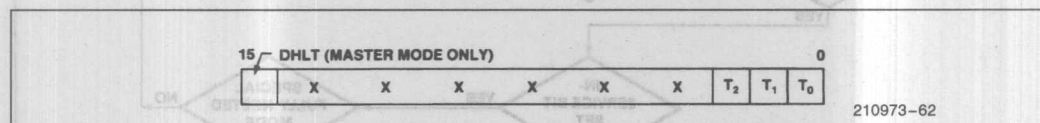


Figure 55. 80186 Interrupt Status Register Format

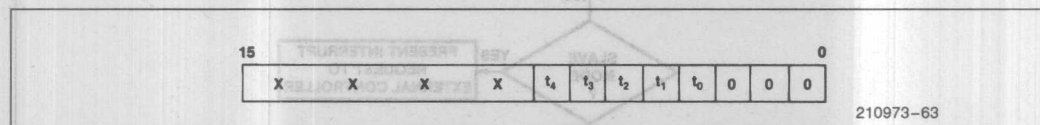


Figure 56. 80186 Interrupt Vector Register Format (Slave Mode only)

6.4 Interrupt Sources

The 80186 interrupt controller receives and arbitrates among many different interrupt request sources, both internal and external. Internal interrupts are processed by the interrupt controller in either Master Mode or Slave Mode. External interrupts are processed by the integrated interrupt controller only in Master Mode. Each interrupt source may be programmed to be a different priority level. An interrupt request generation flow chart is shown in Figure 57. This flowchart is followed independently by each interrupt source.

6.4.1 INTERNAL INTERRUPT SOURCES

The internal interrupt sources are the three timers and the two DMA channels. An interrupt from each of these interrupt sources is latched in the interrupt controller. The state of the pending interrupt can be obtained by reading the interrupt request register. Also,

latched DMA interrupts can be reset by the processor by writing to the interrupt request register. Note that all timers share a common bit in the interrupt request register in master mode. The interrupt controller status register may be read to determine which timer is actually causing the interrupt request. Each timer has a unique interrupt vector (see Section 6.5.1). Thus polling is not required to determine which timer has caused the interrupt in the interrupt service routine. Also, because the timers share a common interrupt control register, they are placed at a common priority level relative to other interrupt sources. Among themselves they have a fixed priority, with timer 0 as the highest priority timer and timer 2 as the lower priority timer.

6.4.2 EXTERNAL INTERRUPT SOURCES

The 80186 interrupt controller will accept external interrupt requests only when it is programmed in Master Mode. In this mode, the external pins associated

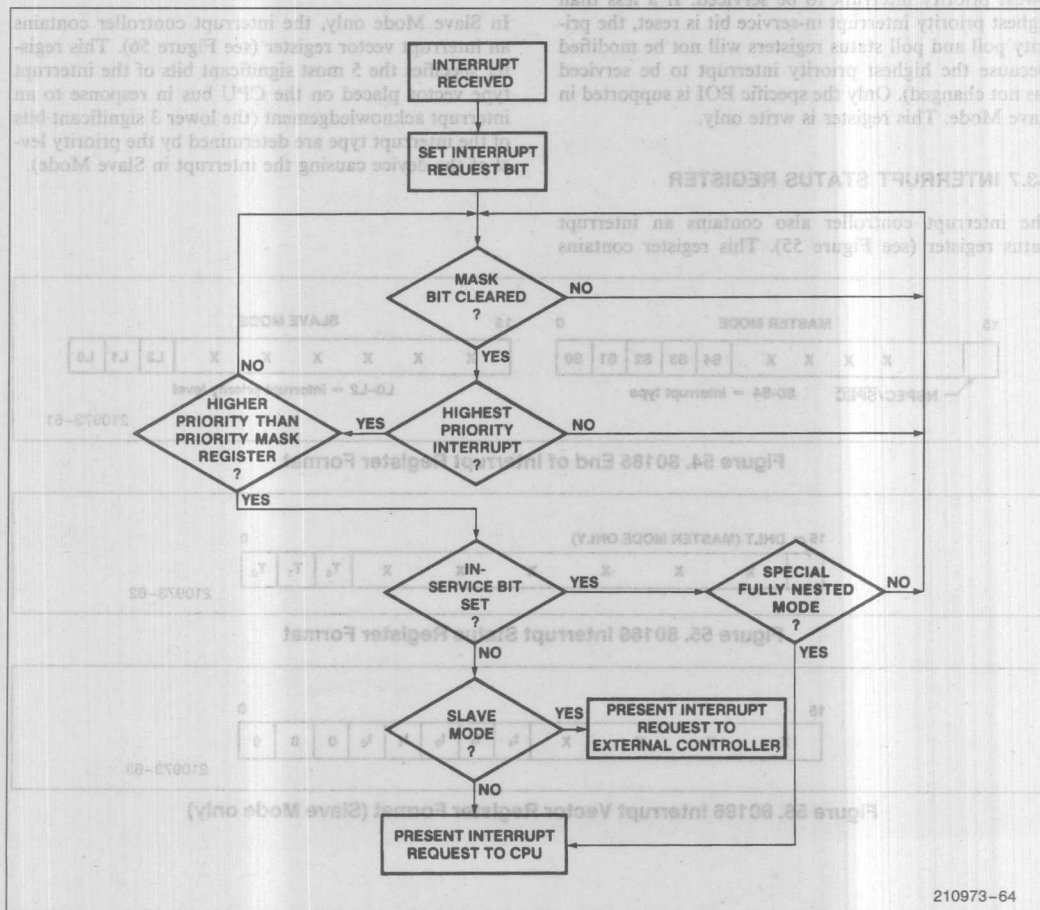


Figure 57. 80186 Interrupt Request Sequencing

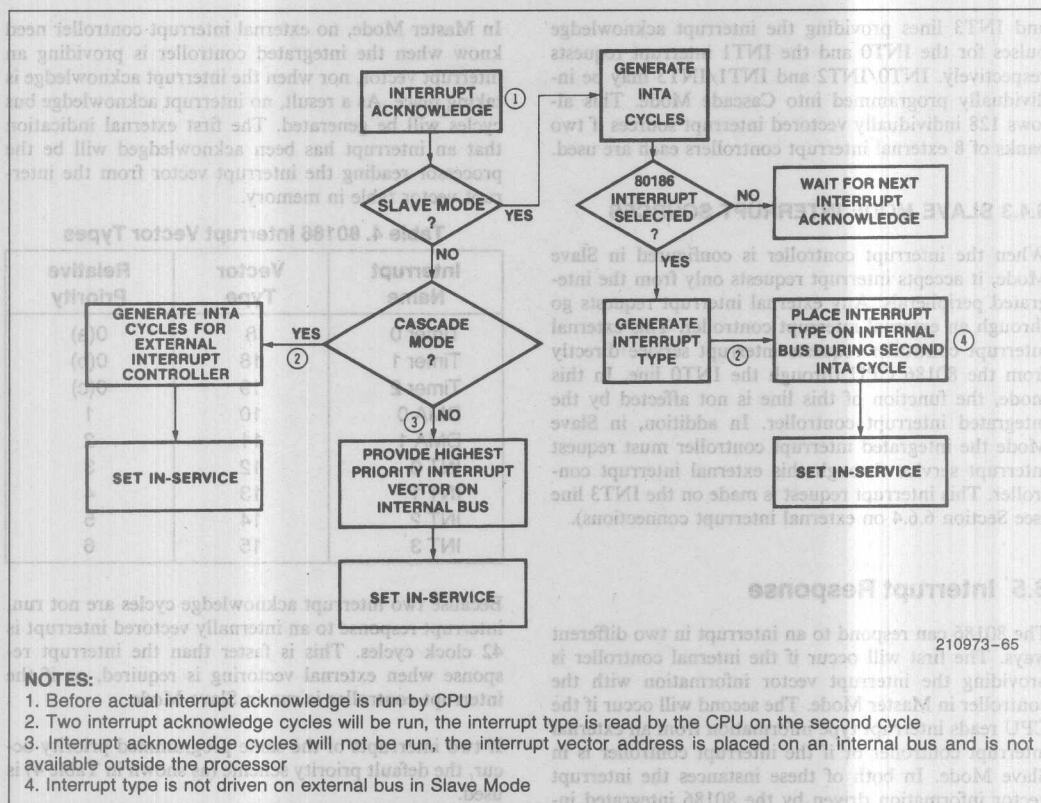


Figure 58. 80186 Interrupt Acknowledge Sequencing

with the interrupt controller may serve either as direct interrupt inputs, or as cascaded interrupt inputs from other interrupt controllers as a programmed option. These options are selected by programming the C and SFNM bits in the INT0 and INT1 control registers (see Figure 50).

When programmed as direct interrupt inputs, the four interrupt inputs are each controlled by an individual interrupt control register. As stated earlier, these registers contain 3 bits which select the priority level for the interrupt and a single bit which enables the interrupt source to the processor. In addition, each of these control registers contains a bit which selects edge or level triggered mode for the interrupt input. When edge triggered mode is selected, a low-to-high transition must occur on the interrupt input before an interrupt is generated, while in level triggered mode, only a high level needs to be maintained to generate an interrupt. In edge triggered mode, the input must remain low at least 1

clock cycle before the input is rearmed. In both modes, the interrupt level must remain high until the interrupt is acknowledged, i.e., the interrupt request is not latched in the interrupt controller. The status of the interrupt input can be shown by reading the interrupt request register. Each of the external pins has a bit in this register which indicates an interrupt request on the particular pin. Note that since interrupt requests on these inputs are not latched by the interrupt controller, if the external input goes inactive, the interrupt requests (and also the bit in the interrupt request register) will also go inactive (low).

If the C (Cascade) bit of the INT0 or INT1 control registers is set, the interrupt input is cascaded to an external interrupt controller. In this mode, whenever the interrupt presented to the INT0 or INT1 line is acknowledged, the integrated interrupt controller will not provide the interrupt type for the interrupt. Instead, two INTA bus cycles will be run, with the INT2

and INT3 lines providing the interrupt acknowledge pulses for the INT0 and the INT1 interrupt requests respectively. INT0/INT2 and INT1/INT3 may be individually programmed into Cascade Mode. This allows 128 individually vectored interrupt sources if two banks of 8 external interrupt controllers each are used.

6.4.3 SLAVE MODE INTERRUPT SOURCES

When the interrupt controller is configured in Slave Mode, it accepts interrupt requests only from the integrated peripherals. Any external interrupt requests go through an external interrupt controller. This external interrupt controller requests interrupt service directly from the 80186 CPU through the INT0 line. In this mode, the function of this line is not affected by the integrated interrupt controller. In addition, in Slave Mode the integrated interrupt controller must request interrupt service through this external interrupt controller. This interrupt request is made on the INT3 line (see Section 6.6.4 on external interrupt connections).

6.5 Interrupt Response

The 80186 can respond to an interrupt in two different ways. The first will occur if the internal controller is providing the interrupt vector information with the controller in Master Mode. The second will occur if the CPU reads interrupt type information from an external interrupt controller or if the interrupt controller is in Slave Mode. In both of these instances the interrupt vector information driven by the 80186 integrated interrupt controller is not available outside the 80186 microprocessor.

In each interrupt mode, when the integrated interrupt controller receives an interrupt response, the interrupt controller will automatically set the in-service bit and reset the interrupt request bit. In addition, unless the interrupt control register for the interrupt is set in Special Fully Nested Mode, the interrupt controller will prevent any interrupts from occurring from the same interrupt line until the in-service bit for that line has been cleared.

6.5.1 INTERNAL VECTORING, MASTER MODE

In Master Mode, the interrupt types associated with all the interrupt sources are fixed and unalterable. These interrupt types are given in Table 5. In response to an internal CPU interrupt acknowledge the interrupt controller will generate the vector address rather than the interrupt type. On the 80186 (like the 8086) the interrupt vector address is the interrupt type multiplied by 4.

In Master Mode, no external interrupt controller need know when the integrated controller is providing an interrupt vector, nor when the interrupt acknowledge is taking place. As a result, no interrupt acknowledge bus cycles will be generated. The first external indication that an interrupt has been acknowledged will be the processor reading the interrupt vector from the interrupt vector table in memory.

Table 4. 80186 Interrupt Vector Types

Interrupt Name	Vector Type	Relative Priority
Timer 0	8	0(a)
Timer 1	18	0(b)
Timer 2	19	0(c)
DMA 0	10	1
DMA 1	11	2
INT 0	12	3
INT 1	13	4
INT 2	14	5
INT 3	15	6

Because two interrupt acknowledge cycles are not run, interrupt response to an internally vectored interrupt is 42 clock cycles. This is faster than the interrupt response when external vectoring is required, or if the interrupt controller is run in Slave Mode.

If two interrupts of the same programmed priority occur, the default priority scheme (as shown in Table 4) is used.

6.5.2 INTERNAL VECTORING, SLAVE MODE

In Slave Mode, the interrupt types associated with the various interrupt sources are alterable. The upper 5 most significant bits are taken from the interrupt vector register, and the lower 3 significant bits are taken from the priority level of the device causing the interrupt. Because the interrupt type, rather than the interrupt vector address, is given by the interrupt controller in this mode the interrupt vector address must be calculated by the CPU before servicing the interrupt.

In Slave Mode, the integrated interrupt controller will present the interrupt type to the CPU in response to the two interrupt acknowledge bus cycles run by the processor. During the first interrupt acknowledge cycle, the external master interrupt controller determines which slave interrupt controller will place its interrupt vector on the microprocessor bus. During the second interrupt acknowledge cycle, the processor reads the interrupt vector from its bus. Thus, these two interrupt acknowl-

age cycles must be run, since the integrated controller will present the interrupt type information only when the external interrupt controller signals the integrated controller that it has the highest pending interrupt request (see Figure 59). The 80186 samples the **SLAVE SELECT** line (INT1) during the falling edge of the clock at the beginning of T_3 of the second interrupt acknowledge cycle. This input must be stable before and after this edge.

These two interrupt acknowledge cycles will be run back to back, and will be **LOCKED** with the **LOCK** output active. The two interrupt acknowledge cycles will always be separated by two idle T states, and wait states will be inserted into the interrupt acknowledge cycle if a ready is not returned by the processor bus interface. The two idle T states are inserted to allow compatibility with an external 8259A interrupt controller.

Because the interrupt acknowledge cycles must be run in Slave Mode and the integrated controller presents an

interrupt type rather than a vector address, the interrupt response time is the same as for an externally vectored interrupt, namely 55 CPU clocks.

6.5.3 EXTERNAL VECTURING

External interrupt vectoring occurs whenever the 80186 interrupt controller is placed in Cascade Mode, Special Fully Nested Mode, or Slave Mode (and the integrated controller is not enabled by the external master interrupt controller). In this mode, the 80186 generates two interrupt acknowledge cycles, reading the interrupt type off the lower 8 bits of the address/data bus on the second interrupt acknowledge cycle (see Figure 59). This interrupt response is exactly the same as the 8086, so that the 8259A interrupt controller can be used exactly as it would in an 8086 system. Notice that the two interrupt acknowledge cycles are **LOCKED**, and that two idle T-states are always inserted between the two interrupt acknowledge bus cycles, and that wait states will be inserted in the interrupt acknowledge cycle if a ready is not returned to the processor. Also

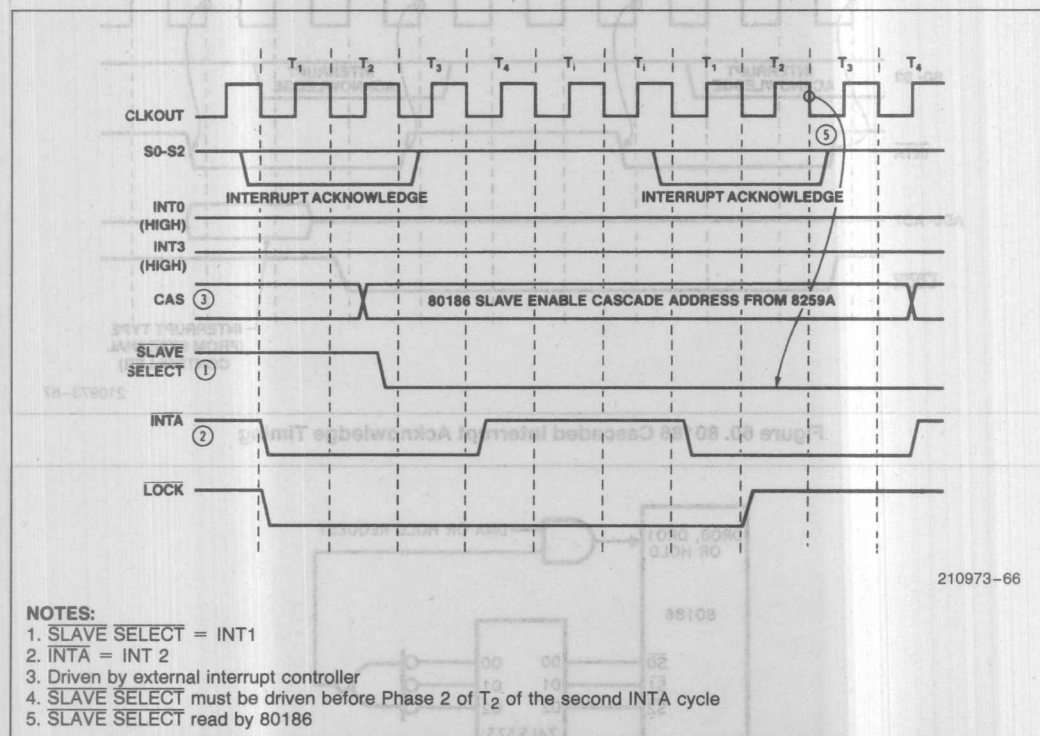


Figure 59. 80186 Slave Mode Interrupt Acknowledge Timing

edge signals, one for interrupts signaled by the INT0 line, and one for interrupts signaled by the INT1 line (on the INT2/INTA0 and INT3/INTA1 lines, respectively). These two interrupt acknowledge signals are mutually exclusive. Interrupt acknowledge status will be driven on the status lines (S0-S2) when either INT2/INTA0 or INT3/INTA1 signal an interrupt acknowledge.

6.5.4 EFFECT OF LOCK PREFIX ON INTERRUPT ACKNOWLEDGE CYCLES

When the interrupt controller is operating in either the cascade or slave modes and an interrupt occurs during an instruction that has been LOCKED by software, the LOCK signal timing shown in Figures 59 and 60 may be altered. Some peripheral devices used with the 80186

require contiguous INTA cycles to allow correct interrupt controller response. In such cases, the external circuitry in Figure 61 should be used to ensure that DMA or HOLD requests are blocked from stealing the bus during INTA cycles.

6.6 Interrupt Controller External Connections

The four interrupt signals can be configured into 3 major options. These are direct interrupt inputs (with the integrated controller providing the interrupt vector), cascaded (with an external interrupt controller providing the interrupt vector), or Slave Mode. In all these modes, any interrupt presented to the external lines must remain set until the interrupt is acknowledged.

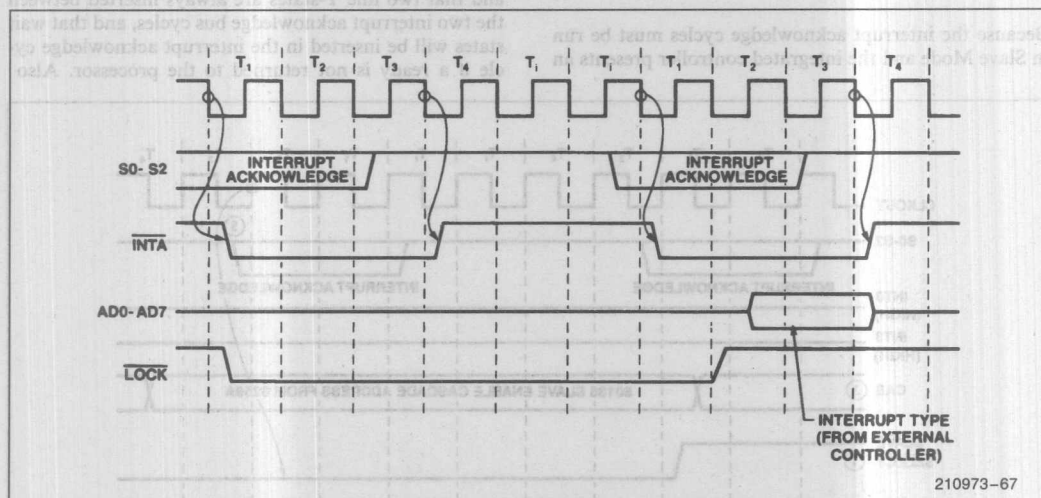


Figure 60. 80186 Cascaded Interrupt Acknowledge Timing

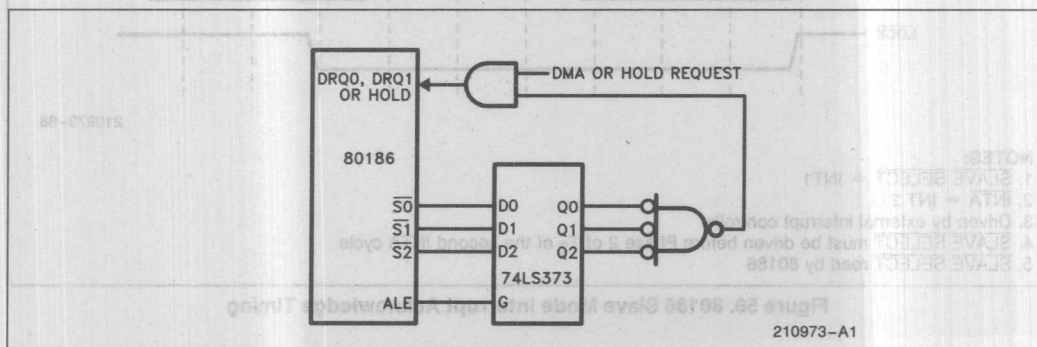


Figure 61. Circuit Blocking DMA or HOLD Request Between INTA Cycles

6.6.1 DIRECT INPUT MODE

When the Cascade Mode bits are cleared, the interrupt input pins are configured as direct interrupt pins (see Figure 62). Whenever an interrupt is received on the input line, the integrated controller will do nothing unless the interrupt is enabled, and it is the highest priority pending interrupt. At this time, the interrupt controller will present the interrupt to the CPU and wait for an interrupt acknowledge. When the acknowledge occurs, it will present the interrupt vector address to the CPU. In this mode, the CPU will not run any external interrupt-acknowledge (INTA) cycles.

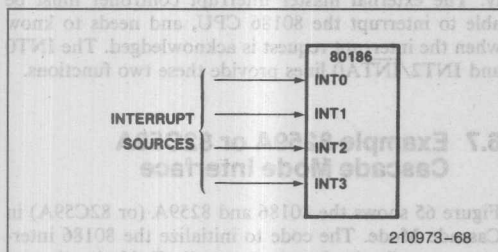


Figure 62. 80186 Non-Cascaded Interrupt Connection

6.6.2 CASCADE MODE

When the Cascade Mode bit is set and the SFNM bit is cleared, the interrupt input lines are configured in Cascade Mode. In this mode, the interrupt input line is paired with an interrupt acknowledge line. The INT2/INTA0 and INT3/INTA1 lines are dual purpose; they can function as direct input lines, or they can function as interrupt acknowledge outputs. INT2/INTA0 provides the interrupt acknowledge for an INT0 input, and INT3/INTA1 provides the interrupt acknowledge for an INT1 input. Figure 63 shows this connection.

When programmed in this mode, in response to an interrupt request on the INT0 line, the 80186 will provide two interrupt acknowledge pulses. These pulses will be provided on the INT2/INTA0 line, and will also be reflected by interrupt acknowledge status being generated on the S0-S2 status lines. The interrupt type will be read on the second pulse. The 80186 externally vectored interrupt response is covered in more detail in Section 6.5.

INT0/INT2/INTA0 and INT1/INT3/INTA1 may be individually programmed into interrupt request/acknowledge pairs, or programmed as direct inputs. This means that INT0/INT2/INTA0 may be programmed as an interrupt/acknowledge pair, while INT1 and INT3/INTA1 each provide separate internally vectored interrupt inputs.

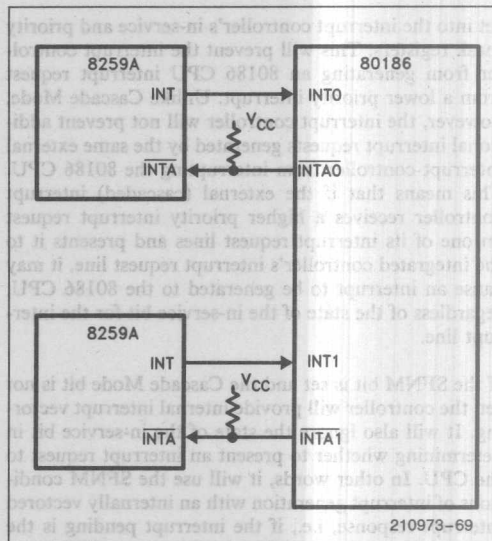


Figure 63. 80186 Cascade and Special Fully Nested Mode Interface

When an interrupt is received on a cascaded interrupt pin, the priority mask bits and the in-service bits in the particular interrupt control register will be set. This will prevent the controller from generating an 80186 CPU interrupt request from a lower priority interrupt. Also, since the in-service bit is set, any subsequent interrupt requests on the particular interrupt input line will not cause the integrated interrupt controller to generate an interrupt request to the 80186 CPU. This means that if the external interrupt controller receives a higher priority interrupt request on one of its interrupt request lines and presents it to the 80186, it will not subsequently be presented to the 80186 CPU by the integrated interrupt controller until the in-service bit for the interrupt line has been cleared.

6.6.3 SPECIAL FULLY NESTED MODE

When both the Cascade Mode bit and the SFNM bit are set, the interrupt input lines are configured in Special Fully Nested Mode. The external interface in this mode is exactly as in Cascade Mode. The only difference is in the conditions allowing an interrupt from the external interrupt controller to the integrated interrupt controller to interrupt the 80186 CPU.

When an interrupt is received from a Special Fully Nested Mode interrupt line, it will interrupt the 80186 CPU if it is the highest priority interrupt pending regardless of the state of the in-service bit for the interrupt source in the interrupt controller. When an interrupt is acknowledged from a special fully nested mode interrupt line, the priority mask bits and the in-service bits in the particular interrupt control register will be

set into the interrupt controller's in-service and priority mask registers. This will prevent the interrupt controller from generating an 80186 CPU interrupt request from a lower priority interrupt. Unlike Cascade Mode, however, the interrupt controller will not prevent additional interrupt requests generated by the same external interrupt controller from interrupting the 80186 CPU. This means that if the external (cascaded) interrupt controller receives a higher priority interrupt request on one of its interrupt request lines and presents it to the integrated controller's interrupt request line, it may cause an interrupt to be generated to the 80186 CPU, regardless of the state of the in-service bit for the interrupt line.

If the SFNM bit is set and the Cascade Mode bit is not set, the controller will provide internal interrupt vectoring. It will also ignore the state of the in-service bit in determining whether to present an interrupt request to the CPU. In other words, it will use the SFNM conditions of interrupt generation with an internally vectored interrupt response, i.e., if the interrupt pending is the highest priority type pending, it will cause a CPU interrupt regardless of the state of the in-service bit for the interrupt. This operation is only applicable to INT0 and INT1, which have SFNM bits in their control registers.

6.6.4 SLAVE MODE

When the SLAVE/MASTER bit in the peripheral relocation register is set, the interrupt controller is in Slave

Mode. In this mode, all four interrupt controller input lines are used to perform the necessary handshaking with the external master interrupt controller. Figure 64 shows the hardware configuration of the 80186 interrupt lines with an external controller in Slave Mode.

Because the integrated interrupt controller is a slave controller, it must be able to generate an interrupt input for an external interrupt controller. It also must be signaled when it has the highest priority pending interrupt to know when to place its interrupt vector on the bus. These two signals are provided by the INT3/Slave Interrupt Output and INT1/Slave Select lines, respectively. The external master interrupt controller must be able to interrupt the 80186 CPU, and needs to know when the interrupt request is acknowledged. The INT0 and INT2/INTA0 lines provide these two functions.

6.7 Example 8259A or 82C59A Cascade Mode Interface

Figure 65 shows the 80186 and 8259A (or 82C59A) in Cascade Mode. The code to initialize the 80186 interrupt controller is given in Appendix E. Notice that an interrupt ready signal must be returned to the 80186 to prevent the generation of wait states in response to the interrupt acknowledge cycles. In this configuration the INT0 and INT2 lines are used as direct interrupt input lines. Thus, this configuration provides 10 external interrupt lines: 2 provided by the 80186 interrupt controller itself, and 8 from the external 8259A. Also, the 8259A is configured as a master interrupt controller. It will only receive interrupt acknowledge pulses in response to an interrupt it has generated. It may be cascaded again to up to 8 additional 8259As (each of which would be configured in Slave Mode).

6.8 Interrupt Latency

Interrupt latency time is the time from when the 80186 receives the interrupt to the time it begins to respond to the interrupt. This is different from interrupt response time, which is the time from when the processor actually begins processing the interrupt to when it actually executes the first instruction of the interrupt service routine. The factors affecting interrupt latency are the instruction being executed and the state of the interrupt enable flip-flop. The interrupt enable flip-flop must be explicitly set by issuing the STI instruction. Since interrupt vectoring automatically clears the flip-flop, it is necessary to set the flip-flop within the interrupt service routine if nested interrupts are desired.

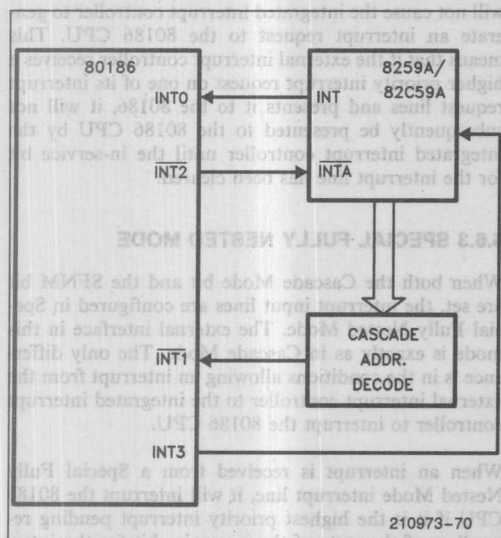


Figure 64. 80186 Slave Mode Interface

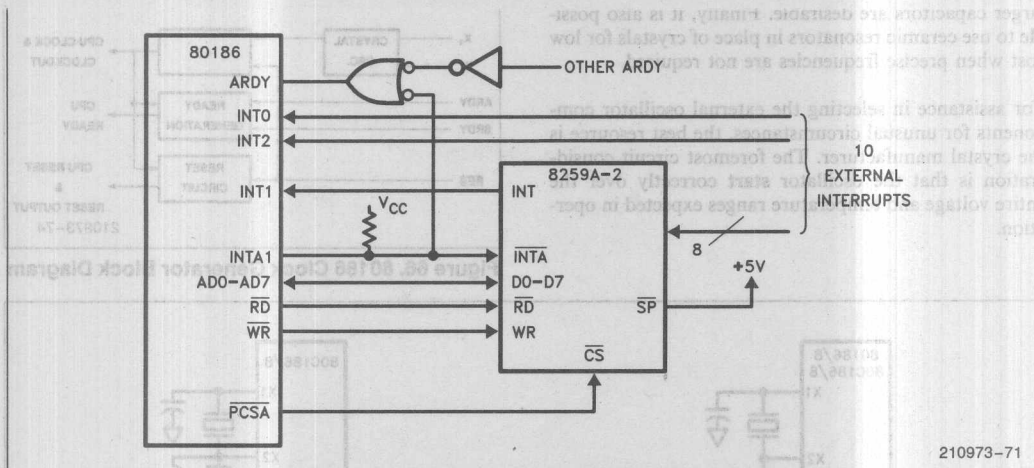


Figure 65. 80186/8259A Interrupt Cascading

When interrupts are enabled in the CPU, the interrupt latency is a function of the instructions being executed. Only repeated instructions will be interrupted before being completed, and those only between their respective iterations. This means that the interrupt latency time could be as long as 69 CPU clocks, which is the time it takes the processor to execute an integer divide instruction (with a segment override prefix, see below), the longest single instruction on the 80186.

Other factors can affect interrupt latency. An interrupt will not be accepted between the execution of a prefix (such as segment override prefixes and lock prefixes) and the instruction. In addition, an interrupt will not be accepted between an instruction which modifies any of the segment registers and the instruction immediately following the instruction. This is required to allow the stack to be changed. If the interrupt were accepted, the return address from the interrupt would be placed on a stack which was not valid (the Stack Segment register would have been modified but the Stack Pointer register would not have been). An interrupt will not be accepted between the execution of the WAIT instruction and the instruction immediately following it if the TEST input is active. If the WAIT sees the TEST input inactive, however, the interrupt will be accepted, and the WAIT will be re-executed after the interrupt return. Finally, the 80C186 and 80C188 will not accept interrupts during refresh bus cycles.

7.0 CLOCK GENERATOR

The 80186 clock generator provides the main clock signal for all 80186 integrated components, and all CPU synchronous devices in the 80186 system. This clock generator includes a crystal oscillator, divide by two counter, reset circuitry, and ready generation logic. A block diagram of the clock generator is shown in Figure 66.

7.1 Crystal Oscillator

All 80186 family microprocessors use a parallel resonant Pierce oscillator. For all NMOS 80186/80188 applications and lower frequency 80C186/80C188 applications, a fundamental mode crystal is appropriate. At higher frequencies, the diminishing thickness of fundamental mode crystals makes a third overtone crystal the appropriate choice. The addition of external capacitors at X1 and X2 is always required, and a third overtone crystal also requires an RC tank circuit to select the third overtone frequency over the fundamental frequency (see Figure 67).

The recommendations given in the 80186 family product data sheets for the values of the external components should be taken only as guidelines since there are situations where the oscillator operation can be modified somewhat. One example would be the case where the circuit layout introduces significant stray capacitance to the X1 and X2 pins. Another example is at low frequencies (CLKOUT less than 6 MHz) where slightly

larger capacitors are desirable. Finally, it is also possible to use ceramic resonators in place of crystals for low cost when precise frequencies are not required.

For assistance in selecting the external oscillator components for unusual circumstances, the best resource is the crystal manufacturer. The foremost circuit consideration is that the oscillator start correctly over the entire voltage and temperature ranges expected in operation.

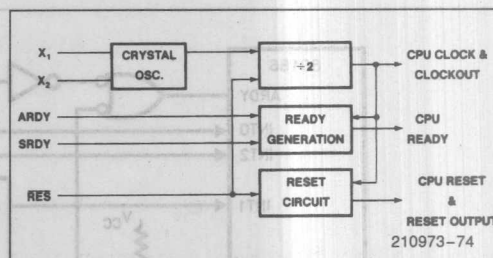


Figure 66. 80186 Clock Generator Block Diagram

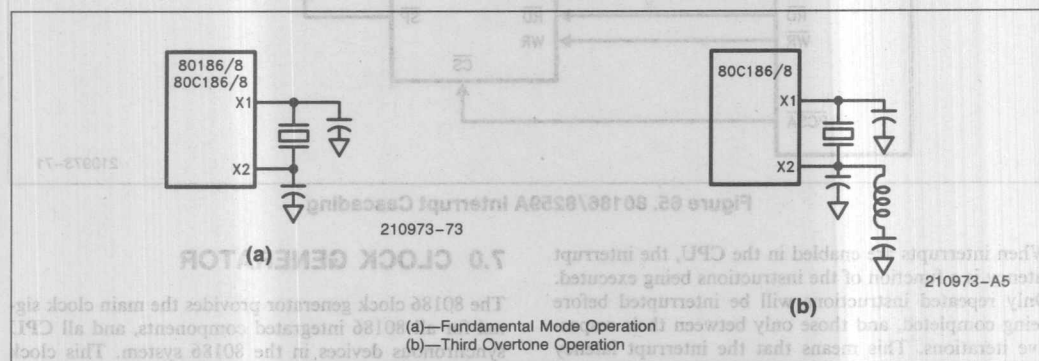


Figure 67. 80186 Family Crystal Connections

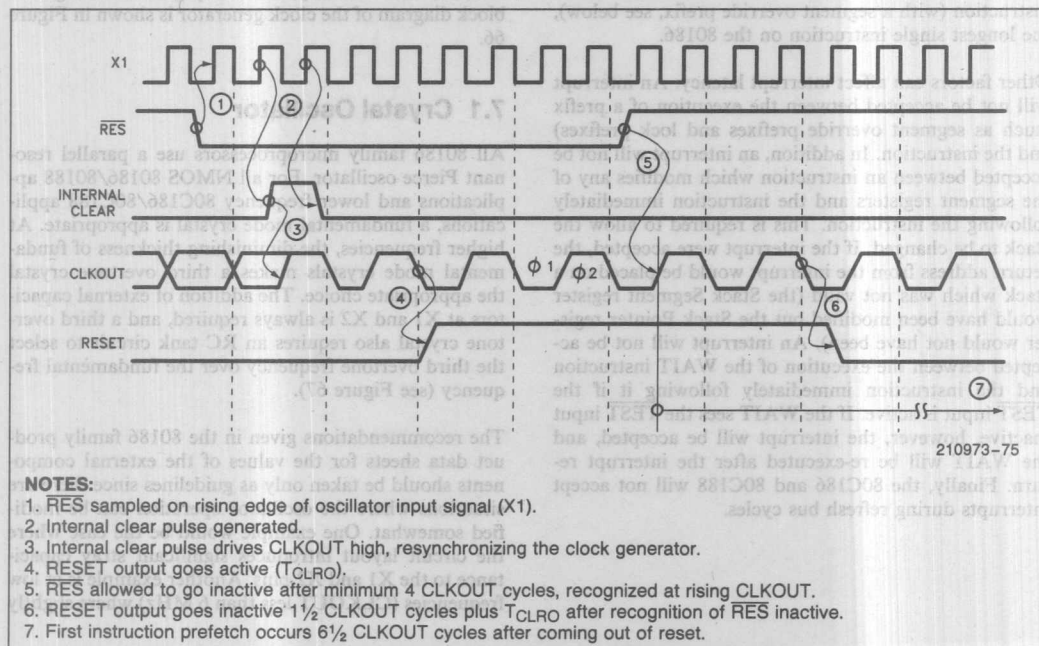


Figure 68. 80186 Clock Generator Reset

7.2 Using an External Oscillator

An external oscillator may be used with the 80186. The external frequency input (EFI) signal is connected directly to the X1 input of the oscillator. X2 should be left not connected. This oscillator input drives an internal divide-by-two counter to generate the CPU clock signal, so the external frequency input can be of practically any duty cycle, so long as the minimum high and low times for the signal (as stated in the data sheet) are met.

7.3 Clock Generator

The output of the crystal oscillator (or the external frequency input) drives a divide by two circuit which generates a 50% duty cycle clock for the 80186 system. All 80186 timing is referenced to this signal, which is available on the CLKOUT pin of the 80186. This signal will change state on the high-to-low transition of the EFI signal.

7.4 Ready Generation

The clock generator also includes the circuitry required for ready generation. Interfacing to the SRDY and ARDY inputs this provides is covered in Section 3.1.6.

7.5 Reset

The 80186 clock generator also provides a synchronized reset signal for the system. This signal is generated from the reset input (RES) to the 80186. The clock generator synchronizes this signal to the clockout signal.

The reset input also resets the divide-by-two counter. A one clock cycle internal clear pulse is generated when the RES input signal goes active. This clear pulse goes active beginning on the first low-to-high transition of the X1 input after RES goes active, and goes inactive on the next low-to-high transition of the X1 input. In

order to insure that the clear pulse is generated on the next oscillator cycle, the RES input signal must satisfy a setup time to the high-to-low oscillator input signal (see Figure 68). During this clear, CLKOUT will be high. On the next high-to-low transition of X1, CLKOUT will go low, and will change state on every subsequent high-to-low X1 transition.

The reset signal presented to the rest of the 80186, and also the signal present on the RESET output pin of the 80186 is synchronized by the high-to-low transition of the clockout signal of the 80186. This signal remains active as long as the RES input also remains active. After the RES input goes inactive, the 80186 will begin to fetch its first instruction (at memory location FFFF0H) after 6 1/2 CPU clock cycles (i.e., T₁ of the first instruction fetch will occur 6 1/2 clock cycles later). To ensure that the RESET output will go inactive on the next CPU clock cycle, the inactive going edge of the RES input must satisfy certain hold and setup times to the low-to-high edge of the CLKOUT signal of the 80186 (see Figure 68).

8.0 CHIP SELECTS

The 80186 includes a chip select unit which provides hardware chip select signals for memory and I/O accesses generated by the 80186 CPU and DMA units. This unit is programmable such that it can fulfill the chip select requirements (in terms of memory device or bank size and speed) of most small and medium sized 80186 systems.

The chip selects are driven only for internally generated bus cycles. Any cycles generated by an external unit (e.g., an external DMA controller) will not cause the chip selects to go active. Thus, any external bus masters must be responsible for their own chip select generation. Also, during a bus HOLD, the 80186 does not float the chip select lines. Therefore, logic must be included to enable the devices which the external bus master wishes to access (see Figure 69).

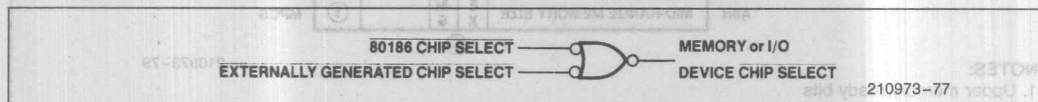


Figure 69. 80186/External Chip Select/Device Chip Select Generation

8.1 Memory Chip Selects

The 80186 provides six discrete memory chip select lines. These signals are named UCS, LCS, and MCS0-3 for Upper Memory Chip Select, Lower Memory Chip Select and Midrange Memory Chip Select 0-3. They are meant (but not limited) to be connected to the three major areas of the 80186 system memory (see Figure 70).

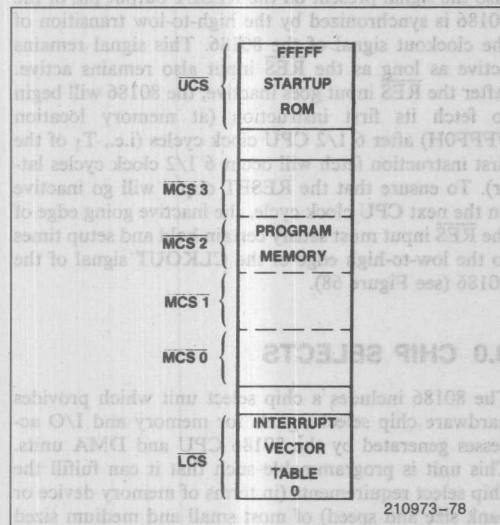


Figure 70. 80186 Memory Areas & Chip Selects

The upper limit of UCS and the lower limit of LCS are fixed at FFFFFH and 00000H in memory space, re-

spectively. The other limit is set by the memory size programmed into the control register for the chip select line. Mid-range memory allows both the base address and the block size of the memory area to be programmed. The only limitation is that the base address must be programmed to be an integer multiple of the total block size. For example, if the block size was 128K bytes (4 32K byte chunks) the base address could be 0 or 20000H, but not 10000H.

The memory chip selects are controlled by 4 registers in the peripheral control block (see Figure 71). These include 1 each for UCS and LCS, the values of which determine the size of the memory blocks addressed by these two lines. The other two registers are used to control the size and base address of the mid-range memory block.

On reset, only UCS is active. It is programmed to be active for the top 1K memory block, to insert 3 wait states to all memory fetches, and to factor external ready for every memory fetch (see Section 8.3 for more information on internal ready generation). None of the other chip select lines will be active until all necessary registers for a signal have been accessed (not necessarily written, a read to an uninitialized register will enable the chip select function controlled by that register).

8.2 Peripheral Chip Selects

The 80186 provides seven discrete chip select lines which are meant to be connected to peripheral components in an 80186 system. Each of these lines is active for one of seven continuous 128 byte areas in memory or I/O space above a programmed base address.

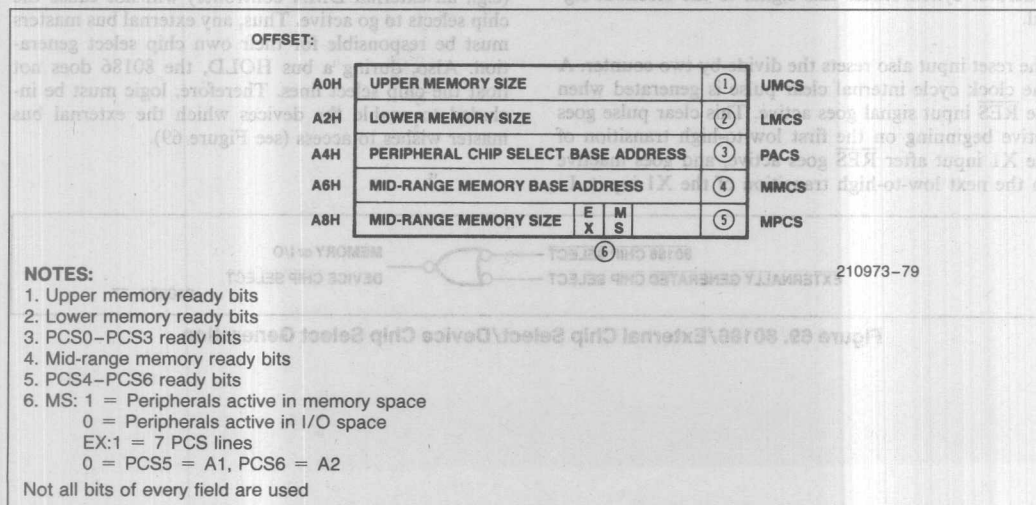


Figure 71. 80186 Chip Select Control Registers

The peripheral chip selects are controlled by two registers in the internal peripheral control block (see Figure 71). These registers set the base address of the peripherals and map the peripherals into memory or I/O space. Both of these registers must be accessed before any of the peripheral chip selects will become active.

A bit in the MPCS register allows $\overline{\text{PCS5}}$ and $\overline{\text{PCS6}}$ to become latched $\overline{\text{A1}}$ and $\overline{\text{A2}}$ outputs. When this option is selected, $\overline{\text{PCS5}}$ and $\overline{\text{PCS6}}$ reflect the state of $\overline{\text{A1}}$ and $\overline{\text{A2}}$ throughout a bus cycle. These allow external peripheral register selection in a system in which the addresses are not latched. Upon reset, these lines are driven high.

8.3 Ready Generation

The 80186 includes a ready generation unit. This unit generates an internal ready signal for all accesses to memory or I/O areas to which the chip select circuitry of the 80186 responds.

For each ready generation area, 0–3 wait states may be inserted by the internal unit. Table 5 shows how the ready control bits should be programmed to provide this. In addition, the ready generation circuit may be programmed to ignore or include the state of the external ready pins. When using both internal and external ready generation, both elements must be fulfilled before a busy cycle will end. The external ready condition is always required upon RESET for accesses involving the top 1K of memory. Therefore, at least one of the ready pins must be connected to functional ready circuitry or be tied HIGH until $\overline{\text{UCS}}$ is reprogrammed early in the initialization sequence.

Table 5. 80186 Wait State Programming

R2	R1	R0	Number of Wait States
0	0	0	0 + external ready
0	0	1	1 + external ready
0	1	0	2 + external ready
0	1	1	3 + external ready
1	0	0	0 (no external ready required)
1	0	1	1 (no external ready required)
1	1	0	2 (no external ready required)
1	1	1	3 (no external ready required)

8.4 Examples of Chip Select Usage

Many examples using the chip select lines are given in the bus interface section of this note (Section 3.2). The key point to remember when using the chip select function is that they are only activated during bus cycles generated by the 80186. When another master has the bus, it must generate its own chip selects. In addition, whenever the bus is given by the 80186 to an external master (through HOLD/HLDA) the 80186 does not float the chip select lines.

8.5 Overlapping Chip Select Areas

Generally, the chip selects of the 80186 should not be programmed such that any two areas overlap. In addition, none of the programmed chip select areas should overlap any locations of the integrated 256-byte control register block. The consequences of doing this are:

Whenever two chip select lines are programmed to respond to the same area, both will be activated during any access to that area. When this is done, the ready bits for both areas **must** be programmed to the same value. If this is not done, the processor response to an access in this area is indeterminate. This rule also applies to overlapping chip selects with the integrated control block.

If any of the chip select areas overlap the integrated 256-byte control block, the timing on the chip select line is altered. An access to the control block will temporarily activate the corresponding chip select pin, but it will go inactive prematurely.

8.6 MCS Functionality and the 80C186

The 80C186 $\overline{\text{MCS0}}$, $\overline{\text{MCS1}}$ and $\overline{\text{MCS3}}$ pins change function when the part is configured for Enhanced Mode (see Section 9.0 for an explanation of Enhanced Mode). The 80C188 $\overline{\text{MCS}}$ pins function the same in both modes. These pins are configured to support an asynchronous numerics floating point coprocessor (see Table 6). Thus, the 80C186 does not provide the complete range of middle chip selects normally available. However, the functionality of the $\overline{\text{MCS2}}$ pin and the programming features of the MPCS and MMCS registers are still available.

Table 6. MCS Pin Definitions

Pin #	Compatible Mode	Enhanced Mode
35	$\overline{\text{MCS3}}$	NPS, Numerics Processor Select
36	$\overline{\text{MCS2}}$	$\overline{\text{MCS2}}$
37	$\overline{\text{MCS1}}$	ERROR, Numerics Processor Error
38	$\overline{\text{MCS0}}$	PEREQ, Processor Extension Request

In Enhanced Mode, it is still possible to program the starting address, block size and ready requirements of the middle chip selects. This allows the user to take advantage of the wait-state generation logic on the 80C186 even though the majority of external chip selects are not active. It is also possible to use $\overline{\text{MCS2}}$ which is active for one fourth the block size (see Figure 72).

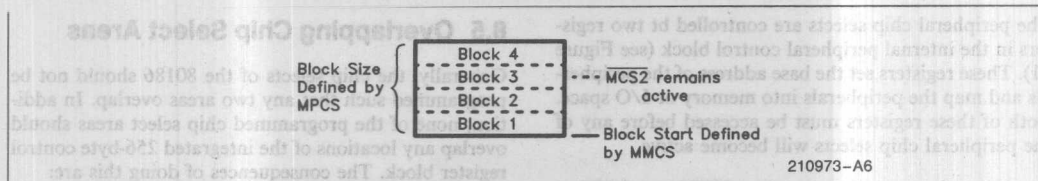


Figure 72. MCS2 Functionality During Enhanced Mode

9.0 80C186 PRODUCT ENHANCEMENTS

The 80C186 and 80C188 are for the most part identical to their NMOS counterparts, and may be used interchangeably. However, aside from the fact that the 80C186 and 80C188 are designed with Intel's CHMOS III technology and provide greater operating frequencies and less power consumption, they also provide two new operating units not found on the 80186 or 80188: the Refresh Control Unit and the Power-Save Unit. To ensure that the new features of the 80C186 are not accidentally programmed in older designs, the 80C186 has two operating modes: Compatible Mode and Enhanced Mode. Compatible Mode implies that the register, programming and pin definition of the 80C186 is identical to that of the 80186. Enhanced Mode implies that the 80C186 provides a super-set of functionality to that of the 80186.

The different modes are selected during RESET. The timing diagram in Figure 73 shows how the 80C186 samples the TEST input pin just before and just after RES is removed to determine if the device will enter Enhanced Mode. Tying the RESET output pin back to

the TEST input pin ensures that the 80C186 or 80C188 enters enhanced mode. If the TEST input is used for external synchronization of code, then RESET can be OR'ed with the other input provided it is always active (low) just after RESET.

When the 80C186 (not the 80C188) is in Enhanced Mode, some of the MCS chip select lines change functionality to support an asynchronous numerics floating-point coprocessor. Refer to Section 8.6 for more detail.

9.1 Refresh Control Unit

To simplify the design of a dynamic memory controller, the 80C186 incorporates integrated address and clock counters which, along with the BIU, facilitate dynamic memory refreshing. A block diagram of the Refresh Control Unit (RCU) and its relationship to the BIU is shown in Figure 74. To the memory interface, a refresh request looks exactly like a memory read bus cycle. This is because a refresh bus cycle is a memory read operation. Because the RCU is integrated into the 80C186, functions such as chip selects and wait-state control can be used effectively.

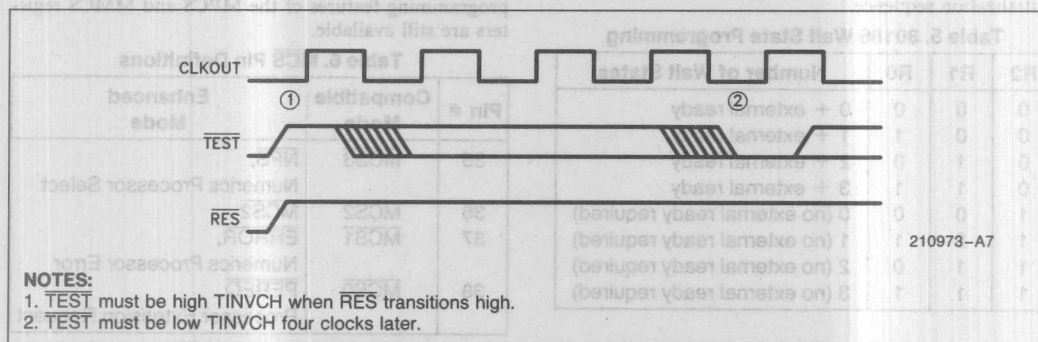


Figure 73. Enhanced Mode Enable Pin Timing

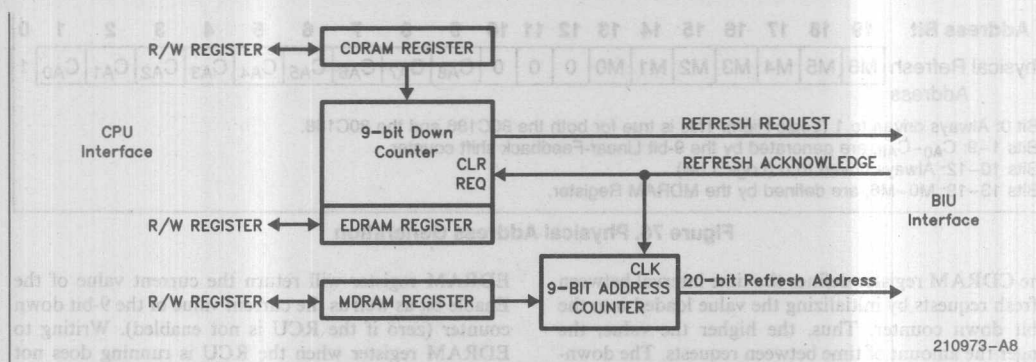


Figure 74. Refresh Control Unit Block Diagram

The 9-bit counter is controlled by the BIU and is used whenever a refresh bus cycle is executed. Thus, any dynamic memory whose refresh address requirement does not exceed nine bits can be directly supported by the 80C186. The 9-bit address counter along with a 6-bit base register define a full 20-bit refresh address. The 9-bit counter generates a signal to initiate a refresh bus cycle. When the counter decrements to 1 (it is decremented every clock cycle), a refresh request is presented to the BIU. When the bus is free, the BIU will run the refresh (memory read) bus cycle. Refresh requests have a higher priority than any other bus request (i.e., CPU, DMA, HOLD).

accessible when the 80C186 or 80C188 are operating in Enhanced mode. Otherwise, a read or write to these registers is ignored.

The three control registers are MDRAM, CDRAM, and EDRAM (see Figure 75). These registers define the operating characteristics of the RCU. The MDRAM register programs the base address (upper 7 bits) of the refresh address (see Figure 76). This allows the refresh address to be mapped into any 4 kilobyte boundary within the 1 megabyte 80C186 address space. The MDRAM register is not altered whenever the refresh address bits (A1 through A9 in Figure 76) roll over. In other words, the refresh address does not act like a linear counter found in a typical DMA controller.

9.1.1 REFRESH CONTROL UNIT PROGRAMMING

There are several registers in the Peripheral Control Block that control the RCU. These registers are only

[illegible]

NOTES:

1. Bits 0–8: T0–T8, Refresh request down counter clock count. These bits are read only and represent the current value of the counter. Any write operation to these bits are ignored.
- Bit 15: E, enables the operation of the refresh control unit.
2. Bits 0–8: C0–C8, define the number of CLKOUT cycles between each refresh request.
3. Bits 9–15: M0–M6, are used to define address bits A13–A19 (respectively) of the 20-bit memory address. These bits are set to zero on RESET.

Figure 75. Refresh Control Unit Registers

Address Bit	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Physical Refresh Address	M6	M5	M4	M3	M2	M1	M0	0	0	0	C _{A8}	C _{A7}	C _{A6}	C _{A5}	C _{A4}	C _{A3}	C _{A2}	C _{A1}	C _{A0}	1

Bit 0: Always driven to 1 (Logic High). This is true for both the 80C186 and the 80C188.
 Bits 1–9: C_{A0}–C_{A8}, are generated by the 9-bit Linear-Feedback shift counter.
 Bits 10–12: Always driven to 0 (Logic Low).
 Bits 13–19: M0–M6, are defined by the MDRAM Register.

Figure 76. Physical Address Generation

The CDRAM register defines the time interval between refresh requests by initializing the value loaded into the 9-bit down counter. Thus, the higher the value, the longer the amount of time between requests. The down-counter is decremented every falling edge of CLKOUT, regardless of the activity of the CPU or BIU. When the counter decrements to 1, a request is generated and the counter is again loaded with the value in the CDRAM register. The amount of time between refresh requests can be calculated using the equation shown in Figure 77. The minimum value that can be programmed into the CDRAM register is 18 (12H) regardless of the operating frequency. This is due to the minimum number of clocks required between each successive request to ensure the BIU has enough time to execute the refresh bus cycle. The BIU is not capable of queuing requests; if another request is generated before the current request is executed, the current request is lost. This applies only to the request itself, not the address associated to the request. The refresh address is only changed after the BIU has run the bus cycle. Thus it is possible to miss refresh requests, but not refresh addresses.

The EDRAM register has two functions, depending on whether it is being written or read. During writes to the EDRAM register, only the Enable bit is active. Setting the Enable bit enables the RCU while clearing the Enable bit disables the RCU. Whenever the RCU is enabled, the contents of the CDRAM register are loaded into the 9-bit down counter and refresh requests will be generated when the counter reaches 1. Disabling the RCU stops and clears the counter. A read of the

EDRAM register will return the current value of the Enable bit as well as the current value of the 9-bit down counter (zero if the RCU is not enabled). Writing to EDRAM register when the RCU is running does not modify the count value in the 9-bit counter.

9.1.2 REFRESH CONTROL UNIT OPERATION

Figure 78 illustrates the two major functions of the refresh control unit that are responsible for initiating and controlling the refresh bus cycles.

The down counter is loaded on the falling edge of CLKOUT, when either the Enable bit is set or the counter decrements to 1. Once loaded, the down counter will decrement every falling edge of CLKOUT (as long as the Enable bit remains set).

When the counter decrements to 1, two things happen. First, a request is generated to the BIU to run a refresh bus cycle. The request remains active until the bus cycle is run. Second, the down counter is reloaded with the value contained in the CDRAM register. At this time, the down counter will again begin counting down every clock cycle. It does not wait until the request has been serviced. This is done to ensure that each refresh request occurs at the correct interval. Otherwise, the time between refresh requests would also be a function of bus activity, which is unpredictable. When the BIU services the refresh request, it will clear the request and increment the refresh address.

$$\frac{R_{\text{PERIOD}} (\mu\text{s}) * \text{FREQ} (\text{MHz})}{\# \text{ Refresh Rows} + (\# \text{ Refresh Rows} * \% \text{ Overhead})} = \text{CDRAM Register Value}$$

R_{period} = Maximum Refresh period specified by the DRAM manufacturer (time in microseconds).
 FREQ = Operating Frequency at 80C186 in MHz.
 # Refresh Rows = Total number of rows to be refreshed.
 % Overhead = Derating factor to compensate for missed refresh requests (typically 1–5%).

Figure 77. Equation to Calculate Refresh Interval

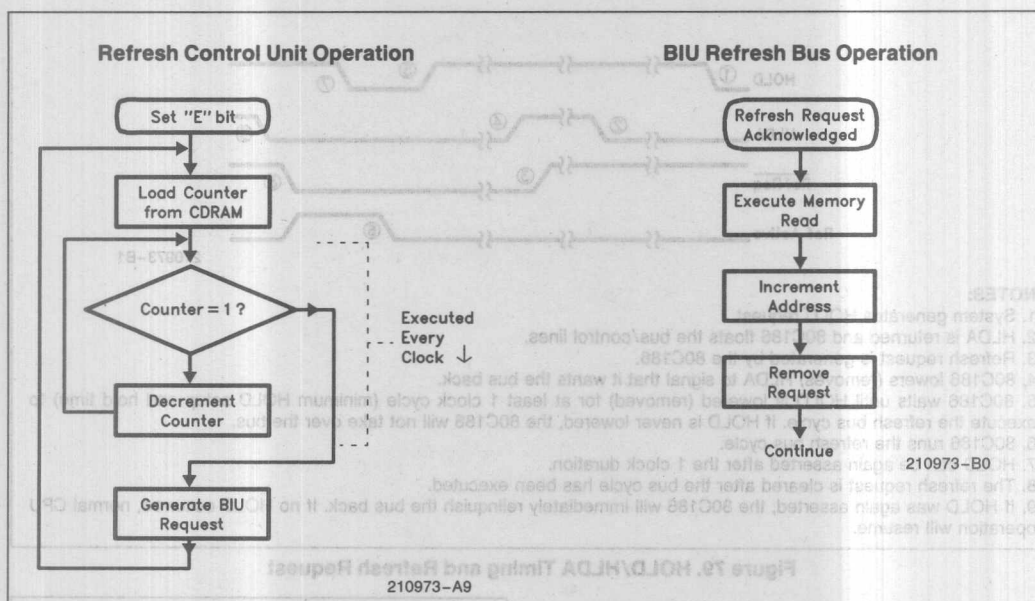


Figure 78. Flowchart of RCU Operation

9.1.3 REFRESH ADDRESS CONSIDERATIONS

The physical address that is generated during a refresh bus cycle is shown in Figure 76, and applies to both the 80C186 and 80C188. The refresh address bits CA_0 through CA_8 are generated using a linear-feedback shift counter which does not increment the addresses linearly from 0 through 1FFF (although they do follow a predictable algorithm). Further, note that for the 80C188, address bit A0 does not toggle during refresh operation, which means that it cannot be used as part of the refresh address applied to the dynamic memory device. Typically, A0 is used as part of memory decoding in 80C188 applications, unlike the 80C186 which uses A0 along with \overline{BHE} to select an upper or lower bank. Therefore, when designing with the 80C188, it is important not to include A0 as part of the row address that is used for refreshing. Appendix K illustrates memory address multiplexing techniques that can be applied to the 80C186 and 80C188.

9.1.4 REFRESH OPERATION AND BUS HOLD

When another bus master has control of the bus, the HLDA signal is kept active as long as the HOLD input remains active. If a refresh request is generated while HOLD is active, the 80C186 will remove (drive inactive) the HLDA signal to indicate to the current bus master that the 80C186 wishes to regain control of the bus (see Figure 79). Only when the HOLD input is removed will the BIU begin the refresh bus cycle.

Therefore, it is the responsibility of the system designer to ensure that the 80C186 can regain the bus if a refresh request is signalled. The sequence of HLDA going inactive while HOLD is active can be used to signal a pending refresh request. HOLD need only go inactive for one clock period to allow the refresh bus cycle to be run. If HOLD is again asserted, the 80C186 will give up the bus after the refresh bus cycle has been run (provided there is not another refresh request generated during that time).

9.2 Power-Save Unit

The Power-Save Unit is intended to benefit applications by lower power consumption while maintaining regular operation of the CPU. The 80C186 Power-Save mechanism lowers current needs by reducing the operating frequency.

The Power-Save Unit is an internal clock divider as shown in Figure 80. Because the Power-Save Unit will change the internal operating frequency, all other units within the 80C186 will be affected by the clock change. This includes the CPU, Timers, Refresh, DMA, and BIU. Thus, by using the Power-Save feature, the net effect is similar to changing the input clock frequency.

PDCON																F1 F0	
Offset F0H																	
NOTE: bit 15 When executing more than one instruction, the PDCON register is programmed to be active for all memory accesses within the top 1K byte of the second LOCKED instruction.																	
Bits 0-1: Clock Divisor Select F1 F0 Division Factor Minimum X1 Frequency 0 0 divide by 1 1 MHz 0 1 divide by 4 4 MHz 1 0 divide by 8 8 MHz 1 1 divide by 16 16 MHz																	
Bits 2-14: Reserved, read back as zero Bit 15: Enable (set) Power-Save mode. Cleared (disabled) on RESET.																	

Figure 81. Power-Save Register Definition

The F0 and F1 bits determine the divisor of the Power-Save unit. Figure 81 provides a list of the various combinations of the bits and their division factor. Note that the divisor is related to the output clock, not the input clock at pin X1. Selecting a divisor of 1 does not reduce the power consumption. The operating clock of the 80C186 must not be divided below the minimum operating frequency specified in data sheet (500 kHz). Figure 81 also indicates the minimum operating frequency required in order to use a specific divisor.

9.2.2 POWER-SAVE OPERATION

When the Enable bit in the PDCON register is set, the clock divider circuitry will turn on during the write to the PDCON register (refer to Figure 82). At the falling edge of T₃ of the register write, CLKOUT will change to reflect the new divisor. If any values of F0-F1 other than zero have been programmed, the CLKOUT period will be increased over undivided CLKOUT, starting with the low phase. CLKOUT will not glitch.

The Power-Save Unit remains active until one of three events happens: either the Enable bit in the PDCON register is cleared, new values for F0 and F1 are programmed, or an unmasked interrupt is received. In the first two cases, the changes directly follow Figure 82.

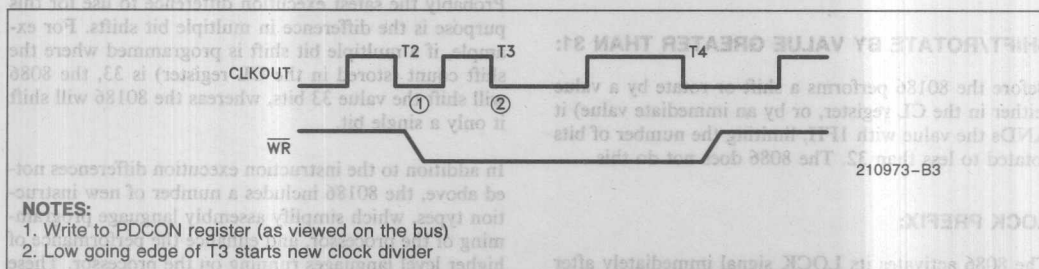


Figure 82. Power-Save Clock Transition

When an unmasked interrupt is received, the operating frequency is changed as shown in Figure 82, but may occur at any T₃ bus state in progress at the time of the interrupt. Thus, it is not possible to determine exactly when, in the event of an interrupt, the Power-Save unit will be disabled.

10.0 SOFTWARE IN AN 80186 SYSTEM

Since the 80186 is object code compatible with the 8086 and 8088, the software in an 80186 system is very similar to that in an 8086 system. Because of the hardware chip select functions, however, a certain amount of initialization code must be included when using those functions on the 80186.

10.1 System Initialization in an 80186 System

The 80186 includes circuitry which directly affects the ability of the system to address memory and I/O devices, namely the chip select circuitry. This circuitry must be initialized before the memory areas and peripheral devices addressed by the chip select signals can be used.

Upon reset, the UMCS register is programmed to be active for all memory fetches within the top 1K byte of memory space. It is also programmed to insert three wait states to all memory accesses within this space. If the hardware chip selects are used, they must be programmed before the processor leaves this 1K byte area of memory. If a jump to an area for which the chips are not selected occurs the processor will fetch garbage. Appendix F shows a typical initialization sequence for the 80186 chip select unit.

10.2 Instruction Execution Differences between the 8086 and 80186

There are a few instruction execution differences between the 8086 and the 80186. These differences are:

UNDEFINED OPCODES:

When the opcodes 63H, 64H, 65H, 66H, 67H, F1H, FEH XX111XXXB and FFH XX111XXXB are executed, the 80186 will execute an illegal instruction exception, interrupt type 6. The 8086 will ignore the opcode.

0FH OPCODE:

When the opcode 0FH is encountered, the 8086 will execute a POP CS, while the 80186 will execute an illegal instruction exception, interrupt type 6.

WORD WRITE AT OFFSET FFFFH:

When a word write is performed at offset FFFFH in a segment, the 8086 will write one byte at offset FFFFH, and the other at offset 0, while the 80186 will write one byte at offset FFFFH, and the other at offset 10000H (one byte beyond the end of the segment). One byte segment underflow will also occur (on the 80186) if a stack PUSH is executed and the Stack Pointer contains the value 1.

SHIFT/ROTATE BY VALUE GREATER THAN 31:

Before the 80186 performs a shift or rotate by a value (either in the CL register, or by an immediate value) it ANDs the value with 1FH, limiting the number of bits rotated to less than 32. The 8086 does not do this.

LOCK PREFIX:

The 8086 activates its LOCK signal immediately after executing the LOCK prefix. The 80186 does not activate the LOCK signal until the processor is ready to begin the data cycles associated with the LOCKed instruction.

NOTE:

When executing more than one LOCKed instruction, always make sure there are 6 bytes of code between the end of the first LOCKed instruction and the start of the second LOCKed instruction.

INTERRUPTED STRING MOVE INSTRUCTIONS:

If an 8086 is interrupted during the execution of a repeated string move instruction, the return value it will push on the stack will point to the last prefix instruction before the string move instruction. If the instruction had more than one prefix (e.g., a segment override prefix in addition to the repeat prefix), it will not be re-executed upon returning from the interrupt. The 80186 will push the value of the first prefix to the repeated instruction, so long as prefixes are not repeated, allowing the string instruction to properly resume.

CONDITIONS CAUSING DIVIDE ERROR WITH AN INTEGER DIVIDE:

The 8086 will cause a divide error whenever the absolute value of the quotient is greater than 7FFFH (for word operations) or if the absolute value of the quotient is greater than 7FH (for byte operations). The 80186 has expanded the range of negative numbers allowed as a quotient by 1 to include 8000H and 80H. These numbers represent the most negative numbers representable using 2's complement arithmetic (equaling -32768 and -128 in decimal, respectively).

ESC OPCODE:

The 80186 may be programmed to cause an interrupt type 7 whenever an ESCape instruction (used for coprocessors like the 8087) is executed. The 8086 has no such provision. Before the 80186 performs this trap, it must be programmed to do so.

These differences can be used to determine whether the program is being executed on an 8086 or an 80186. Probably the safest execution difference to use for this purpose is the difference in multiple bit shifts. For example, if a multiple bit shift is programmed where the shift count (stored in the CL register) is 33, the 8086 will shift the value 33 bits, whereas the 80186 will shift it only a single bit.

In addition to the instruction execution differences noted above, the 80186 includes a number of new instruction types, which simplify assembly language programming of the processor, and enhance the performance of higher level languages running on the processor. These new instructions are covered in depth in the 8086/80186 users manual and in Appendix H of this note.

APPENDIX A PERIPHERAL CONTROL BLOCK

All the integrated peripherals within the 80186 microprocessor are controlled by sets of registers contained within an integrated peripheral control block. The registers are physically located within the peripheral devices they control, but are addressed as a single block of registers. This set of registers encompasses 256 contiguous bytes and can be located on any 256 byte boundary of the 80186 memory or I/O space. Maps of these registers are shown in Figure A-1 for the 80186/80188 and in Figure A-2 for the 80C186/80C188. Any unused bytes are reserved.

A.1 SETTING THE BASE LOCATION OF THE PERIPHERAL CONTROL BLOCK

In addition to the control registers for each of the integrated 80186 peripheral devices, the peripheral control

block contains the peripheral control block relocation register. This register allows the peripheral control block to be re-located on any 256 byte boundary within the processor's memory or I/O space. Figure A-2 shows the layout of this register.

This register is located at offset FEH within the peripheral control block. Since it is itself contained within the peripheral control block, any time the location of the peripheral control block is moved, the location of the relocation registers will also move.

In addition to the peripheral control block relocation information, the relocation register contains two additional bits. One is used to set the interrupt controller into slave mode. The other is used to force the processor to trap whenever an ESCape (coprocessor) instruction is encountered.

	OFFSET
Relocation Register	FEH
DMA Descriptors Channel 1	DAH D0H
DMA Descriptors Channel 0	CAH C0H
Chip-Select Control Registers	A8H A0H
Timer 2 Control Registers	66H 60H 5EH
Timer 1 Control Registers	58H 56H
Timer 0 Control Registers	50H
Interrupt Controller Registers	3EH 20H

Figure A-1. 80186/80188 Integrated Peripheral Control Block

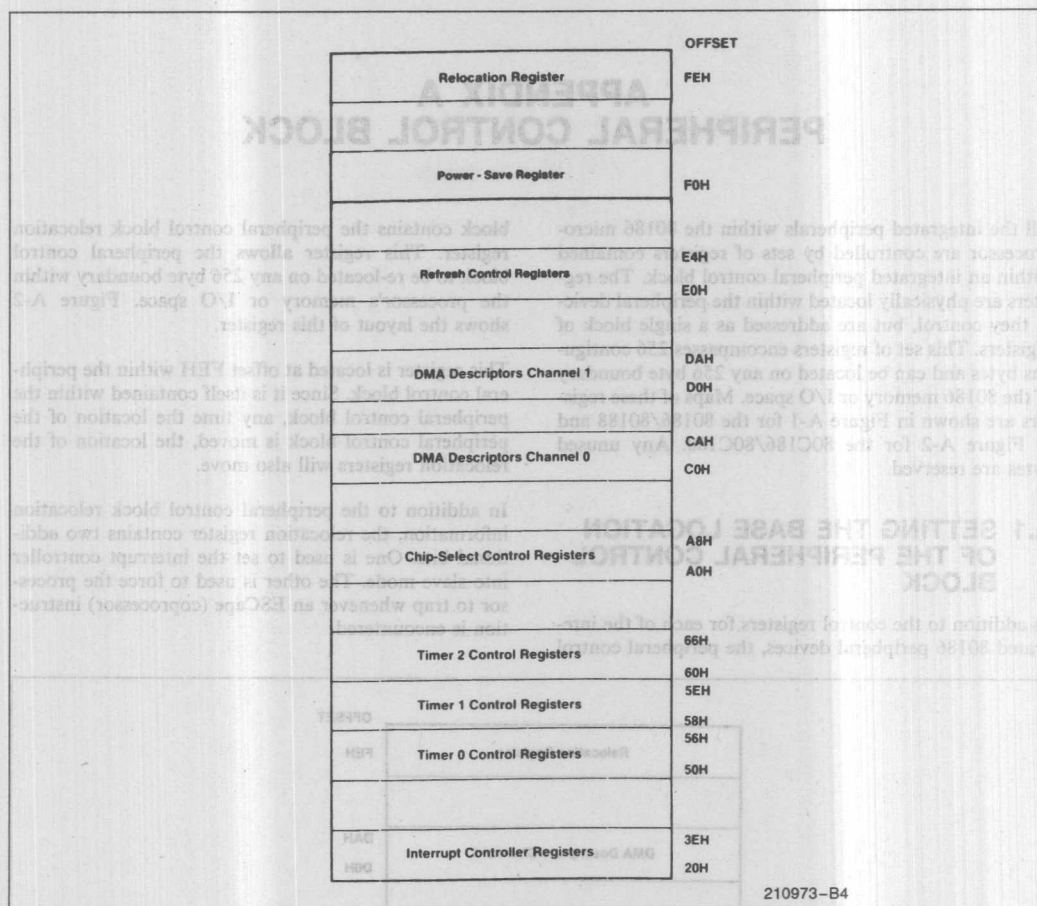


Figure A-2. 80C186/80C188 Integrated Peripheral Control Block

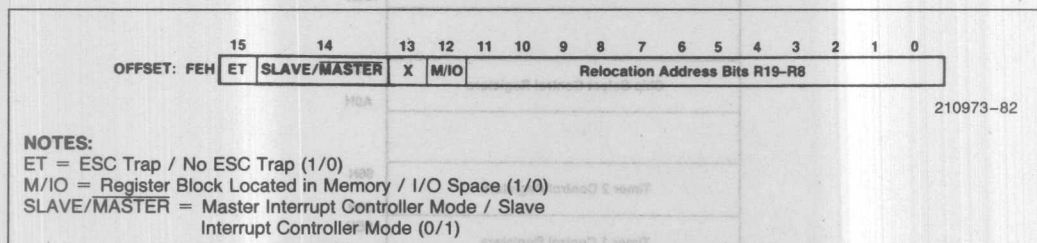


Figure A-3. 80186 Relocation Register Layout

Because the relocation register is contained within the peripheral control block, upon reset the relocation register is automatically programmed with the value 20FFH. This means that the peripheral control block will be located at the very top (FF00H to FFFFH) of I/O space. Thus, after reset the relocation register will be located at word location FFFE H in I/O space.

To relocate the peripheral control block to the memory range 10000H-100FFH, for example, the user programs the relocation register with the value 1100H. Since the relocation register is contained within the peripheral control block, it moves to word location 100FEH in memory space.

Whenever mapping the 188 peripheral control block to another location, the programming of the relocation register should be done with a byte write (i.e., OUT DX,AL). Any access to the control block is done 16 bits at a time. Thus, internally, the relocation register will get written with 16 bits of the AX register while externally, the BIU will run only one 8 bit bus cycle. If a word instruction is used (i.e., OUT DX,AX), the relocation register will be written on the first bus cycle. The BIU will then run a second bus cycle which is unnecessary. The address of the second bus cycle will no longer be within the control block (i.e., the control block was moved on the first cycle), and therefore, will require the generation of an external ready signal to complete the cycle. For this reason we recommend byte operations to the relocation register. Byte instructions may also be used for the other registers in the control block and will eliminate half of the bus cycles required if a word operation had been specified. Byte operations are only valid on even addresses though, and are undefined on odd addresses.

A.2 Peripheral Control Block Registers

Each of the integrated peripherals' control and status registers are located at a fixed location above the programmed base location of the peripheral control block. There are many locations within the peripheral control block which are not assigned to any peripheral. If a write is made to any of these locations, the bus cycle will be run, but the value will not be stored in any internal location. This means that if a subsequent read is made to the same location, the value written will not be read back.

The processor will run an external bus cycle for any memory or I/O cycle which accesses a location within the integrated control block. This means that the address, data, and control information will be driven on

the 80186 external pins just as if a "normal" bus cycle had been run. Any information returned by an external device will be ignored, however, even if the access was to a location which does not correspond to any of the integrated peripheral control registers. The above is also true for the 80188, except that the word access made to the integrated registers will be performed in a single bus cycle internally, while externally, the BIU runs two bus cycles.

The processor internally generates a ready signal whenever any of the integrated peripherals are accessed; thus any external ready signals are ignored. This ready will also be returned if an access is made to a location within the 256 byte area of the peripheral control block which does not correspond to any integrated peripheral control register. The processor will insert 0 wait states to any access within the integrated peripheral control block except for accesses to the timer registers. Any access to the timer control and counting registers will incur 1 wait state. This wait state is required to properly multiplex processor and counter element accesses to the timer control registers.

All accesses made to the integrated peripheral control block will be word accesses. Any write to the integrated registers will modify all 16 bits of the register, whether the opcode specified a byte write or a word write. A byte read from an even location should cause no problems, but the data returned when a byte read is performed from an odd address within the peripheral control block is undefined. This is true both for the 80186 and the 80188. As stated above, even though the 80188 has an external 8 bit data bus, internally it is still a 16-bit machine. Thus, the word accesses performed to the integrated registers by the 80188 will each occur in a single bus cycle internally while externally the BIU runs two bus cycles. The DMA controller cannot be used for either read or write accesses to the peripheral control block.

APPENDIX B 80186 SYNCHRONIZATION INFORMATION

Many input signals to the 80186 are asynchronous, that is, a specified set up or hold time is not required to insure proper functioning of the device. Associated with each of these inputs is a synchronizer which samples this external asynchronous signal, and synchronizes it to the internal 80186 clock.

B.1 WHY SYNCHRONIZERS ARE REQUIRED

Every data latch requires a certain set up and hold time in order to operate properly. At a certain window within the specified set up and hold time, the part will actually try to latch the data. If the input makes a transition within this window, the output will not attain a stable state within the given output delay time. The size of this sampling window is typically much smaller than the actual window specified by the data sheet, however part to part variation could move this window around within the specified window in the data sheet.

Even if the input to a data latch makes a transition while a data latch is attempting to latch this input, the output of the latch will attain a stable state after a certain amount of time, typically much longer than the normal strobe to output delay time. Figure B-1 shows a normal input to output strobed transition and one in which the input signal makes a transition during the latch's sample window. In order to synchronize an asynchronous signal, all one needs to do is to sample the signal into one data latch long enough for the output to stabilize, then latch it into a second data latch. Since the time between the strobe into the first data latch and the strobe into the second data latch allows the first data latch to attain a steady state (or to resolve the asynchronous signal), the second data latch will be presented with an input signal which satisfies any set up and hold time requirements it may have.

Thus, the output of this second latch is a synchronous signal with respect to its strobe input.

A synchronization failure can occur if the synchronizer fails to resolve the asynchronous transition within the

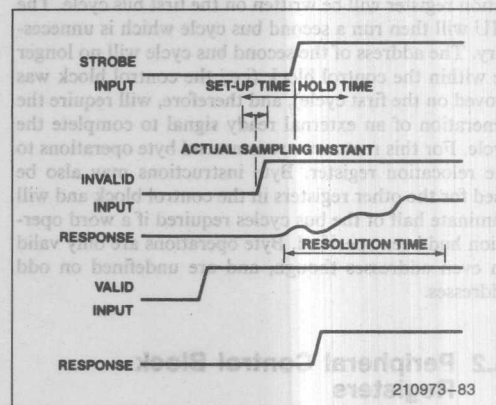


Figure B-1. Valid and Invalid Latch Input Transitions and Responses

time between the two latch's strobe signals. The rate of failure is determined by the actual size of the sampling window of the data latch, and by the amount of time between the strobe signals of the two latches. Obviously, as the sampling window gets smaller, the number of times an asynchronous transition will occur during the sampling window will drop. In addition, however, a smaller sampling window is also indicative of a faster resolution time for an input transition which manages to fall within the sampling window.

B.2 80186 SYNCHRONIZERS

The 80186 contains synchronizers on the \overline{RES} , \overline{TEST} , $TmrIn0-1$, $DRQ0-1$, NMI , $INT0-3$, $ARDY$, and $HOLD$ input lines. Each of these synchronizers use the two stage synchronization technique described above (with some minor modifications for the $ARDY$ line, see section 3.1.6). The sampling window of the latches is designed to be in the tens of pico-seconds, and should allow operation of the synchronizers with a mean time between failures of over 30 years assuming continuous operation.

APPENDIX C

80186 EXAMPLE DMA INTERFACE CODE

```

$mod186
name                assembly.example.80186.DMA_support

; This file contains an example procedure which initializes the 80186 DMA
; controller to perform the DMA transfers between the 80186 system and the
; 8272 Floppy Disk Controller (FDC). It assumes that the 80186
; peripheral control block has not been moved from its reset location.

arg1                equ    word ptr [BP + 4]
arg2                equ    word ptr [BP + 6]
arg3                equ    word ptr [BP + 8]

DMA.FROM.LOWER      equ    0FFC0h
DMA.FROM.UPPER      equ    0FFC2h
DMA.TO.LOWER        equ    0FFC4h
DMA.TO.UPPER        equ    0FFC6h
DMA.COUNT           equ    0FFC8h
DMA.CONTROL         equ    0FFCAh
DMA.TO.DISK.CONTROL equ    01486h

; DMA register locations
; destination synchronization
; source to memory, incremented
; destination to I/O
; no terminal count
; byte transfers

DMA.FROM.DISK.CONTROL equ    0A046h

; source synchronization
; source to I/O
; destination to memory, incr
; no terminal count
; byte transfers

FDC.DMA             equ    6B8h
FDC.DATA            equ    688h
FDC.STATUS          equ    680h

; FDC DMA address
; FDC data register
; FDC status register

cgroup              group    code
code                segment
                    public    set_dma_
                    assume    cs:cgroup

; set_dma (offset,to) programs the DMA channel to point one side to the
; disk DMA address, and the other to memory pointed to by ds:offset. If
; 'to' = 0 then will be a transfer from disk to memory; if
; 'to' = 1 then will be a transfer from memory to disk. The parameters to
; the routine are passed on the stack.

set_dma_            proc    near
                    enter    0,0
                    push    AX
                    push    BX
                    push    DX
                    test    arg2,1
                    jz      from_disk

; performing a transfer from memory to the disk controller

                    mov     AX,DS
                    rol     AX,4

```

210973-84

```

mov     BX,AX                ; save the result...
mov     DX,DMA.FROM.UPPER    ; prgm the upper 4 bits of the
out     DX,AX                ; DMA source register
and     AX,0FFF0h            ; form the lower 16 bits of the
                                ; physical address
add     AX,arg1              ; add the offset
mov     DX,DMA.FROM.LOWER    ; prgm the lower 16 bits of the
out     DX,AX                ; DMA source register
jnc     no.carry.from        ; check for carry out of addition
inc     BX                   ; if carry out, then need to adj
mov     AX,BX                ; the upper 4 bits of the pointer
mov     DX,DMA.FROM.UPPER
out     DX,AX

no.carry.from:
mov     AX,FDC.DMA           ; prgm the low 16 bits of the DMA
mov     DX,DMA.TO.LOWER      ; destination register
out     DX,AX
xor     AX,AX                ; zero the up 4 bits of the DMA
mov     DX,DMA.TO.UPPER      ; destination register
out     DX,AX
mov     AX,DMA.TO.DISK.CONTROL ; prgm the DMA ctl reg
mov     DX,DMA.CONTROL       ; note: DMA may begin immediately
out     DX,AX                ; after this word is output
pop     DX
pop     BX
pop     AX
leave
ret

from.disk:
; performing a transfer from the disk to memory
;
mov     AX,DS
rol     AX,4
mov     DX,DMA.TO.UPPER
out     DX,AX
mov     BX,AX
and     AX,0FFF0h
add     AX,arg1
mov     DX,DMA.TO.LOWER
out     DX,AX
jnc     no.carry.to
inc     BX
mov     AX,BX
mov     DX,DMA.TO.UPPER
out     DX,AX

no.carry.to:
mov     AX,FDC.DMA
mov     DX,DMA.FROM.LOWER
out     DX,AX
xor     AX,AX
mov     DX,DMA.FROM.UPPER
out     DX,AX
mov     AX,DMA.FROM.DISK.CONTROL
mov     DX,DMA.CONTROL
out     DX,AX
pop     DX
pop     BX
pop     AX
leave
ret
endp

set.dma.
code
ends
end
210973-86

```

210973-85

APPENDIX D

80186 EXAMPLE TIMER INTERFACE CODE

```

$mod186
name example.80186.timer.code
;
; this file contains example 80186 timer routines. The first routine
; sets up the timer and interrupt controller to cause the timer
; to generate an interrupt every 10 milliseconds, and to service
; interrupt to implement a real time clock. Timer 2 is used in
; this example because no input or output signals are required.
; The code example assumes that the peripheral control block has
; not been moved from its reset location (FF00-FFFF in I/O space).
;
arg1 equ word ptr [BP + 4]
arg2 equ word ptr [BP + 6]
arg3 equ word ptr [BP + 8]
timer_2int equ 19 ; timer 2 has vector type 19
timer_2control equ 0FF66h
timer_2maxctl equ 0FF62h
timer_intctl equ 0FF32h ; interrupt controller regs
coi_register equ 0FF22h
interruptstat equ 0FF30h

data segment public 'data'
public hour_, minute_, second_, msec_
msec_ db ?
hour_ db ?
minute_ db ?
second_ db ?
data ends

cgroup group code
dgroup data

code segment public 'code'
public settime.
assume cs:code, ds:dgroup
;
; settime(hour, minute, second) sets the time variables, initializes the
; 80186 timer2 to provide interrupts every 10 milliseconds, and
; programs the interrupt vector for timer 2
;
settime. proc near
enter 0, 0 ; set stack addressability
push AX ; save registers used
push DX
push SI
push DS
xor AX, AX ; set the interrupt vector
; the timers have unique
; interrupt
; vectors even though they share
; the same control register

mov DS, AX

mov SI, 4 * timer_2int

```

210973-87

```

mov word ptr DS:[SI],offset timer_2_interrupt_routine
inc SI
inc SI
mov DS:[SI],CS
pop DS

```

```

mov AX,arg1 ; set the time values
mov hour,AL
mov AX,arg2
mov minute,AL
mov AX,arg3
mov second,AL
mov msec,0

```

```

mov DX,timer2.maxctl ; set the max count value
mov AX,20000 ; 10 ms / 500 ns (timer 2 counts)
; at 1/4 the CPU clock rate)
out DX,AX
mov DX,timer2.control ; set the control word
mov AX,1110000000000001b ; enable counting
; generate interrupts on TC
; continuous counting
out DX,AX

```

```

mov DX,timer.intctl ; set up the interrupt controller
mov AX,0000b ; unmask interrupts
; highest priority interrupt
out DX,AX
sti ; enable processor interrupts
pop SI
pop DX
pop AX
leave
ret

```

set.time.

timer2.interruptRoutine

```

proc far
push AX
push DX
cmp msec,99 ; see if one second has passed
jae bump.second ; if above or equal...
msec
reset.intctl

```

bump.second:

```

mov msec,0 ; reset millisecond
cmp second,59 ; see if one minute has passed
jae bump.minute
second
reset.intctl

```

bump.minute:

```

mov second,0
cmp minute,59 ; see if one hour has passed
jae bump.hour
minute
reset.intctl
pop DX
pop AX

```

timer2.interruptRoutine ends
code ends

210973-89

210973-88


```

ret
set.baud.      ret
code          endp
              ends
              end

$mod186
name          example.80186.countCode

;
; this file contains example 80186 timer routines. The third routine
; sets up the timer as an external event counter. In this mode,
; Timer 1 is used to count transitions on its input pin. After
; the timer has been set up by the routine, the number of
; events counted can be directly read from the timer count
; register at location FF58H in I/O space. The timer will
; count a maximum of 65535 timer events before wrapping
; around to zero. This code example also assumes that the
; peripheral control block has not been moved from its reset
; location (FF00-FFFF in I/O space).

timer1.control equ 0FF5Eh
timer1.max.cnt equ 0FF5Ah
timer1.cnt.reg equ 0FF58H

code          segment      public 'code'
              assume cs:code

;
; set_count() initializes the 80186 timer1 as an event counter
;
set_count:
proc near
push AX      ; save registers used
push DX
mov DX,timer1.max.cnt ; set the max count value
mov AX,0      ; allows the timer to count
                ; all the way to FFFFH
out DX,AX
mov DX,timer1.control ; set the control word
mov AX,1100000000000101b ; enable counting
                ; no interrupt on TC
                ; continuous counting
                ; single max count register
                ; external clocking
out DX,AX
xor AX,AX      ; zero AX
mov DX,timer1.cnt.reg ; and zero the count in the timer
out DX,AX      ; count register
pop DX
pop AX
ret
set_count:
code          endp
              ends
              end

```

210973-91

APPENDIX E

80186 EXAMPLE INTERRUPT CONTROLLER INTERFACE CODE

```

; $mod186
; name
; example.80186.interrupt.code
;
; This routine configures the 80186 interrupt controller to provide
; two cascaded interrupt inputs (through an external 8259A
; interrupt controller on pins INT0/INT2) and two direct
; interrupt inputs (on pins INT1 and INT3). The default priority
; levels are used. Because of this, the priority level programmed
; into the control register is set the 111, the level all
; interrupts are programmed to at reset.
;
int0.control      equ    0FF38H
int1.mask         equ    0FF28H
;
; code
; segment
; assume CS:code
; public 'code'
; set.int.
; proc
; near
; push
; push DX
; push AX
;
; mov AX,0100111B
; Cascade Mode
; interrupt unmasked
;
; mov DX,int0.control
; out DX,AX
;
; mov AX,01001101B
; now unmask the other external
; interrupts
;
; mov DX,int1.mask
; out DX,AX
;
; pop AX
; pop DX
; ret
; endp
; ends
; set.int.
; code
;
; $mod186
; name
; example.80186.interrupt.code
;
; This routine configures the 80186 interrupt controller into Slave
; Mode. This code does not initialize any of the 80186
; integrated peripheral control registers, nor does it initialize
; the external 8259A interrupt controller.
;
; relocation.reg      equ    0FFFEH
;
; code
; segment
; assume CS:code
; public 'code'
; set.Slave
; proc
; near
; push
; push DX
; push AX
;
; mov DX,relocation.reg
; in AX,DX
; or AX,0100000000000000B
; read old contents of register
; set the Slave/Master mode bit
;
; out DX,AX

```

210973-92

APPENDIX F

80186/8086 EXAMPLE SYSTEM INITIALIZATION CODE

```

name                example.80186.system.init
:
:   This file contains a system initialization routine for the 80186
:   or the 8086. The code determines whether it is running on
:   an 80186 or an 8086, and if it is running on an 80186, it
:   initializes the integrated chip select registers.
:
restart              segment at 0FFFFh
:   This is the processor reset address at 0FFFF0H
:
:               org 0
:               jmp far ptr initialize
:               ends

initLhw              segment at 0FFFF0h
:               assume CS: initLhw
:
:   This segment initializes the chip selects. It must be located in the
:   top 1K to insure that the ROM remains selected in the 80186
:   system until the proper size of the select area can be programmed.
:
UMCS.reg             equ 0FFA0H           ; chip select register locations
LMCS.reg             equ 0FFA2H
PACS.reg             equ 0FFA4H
MPCS.reg             equ 0FFA8H
UMCS.value           equ 0F038H           ; 64K, no wait states
LMCS.value           equ 07F8H           ; 32K, no wait states
PACS.value           equ 007EH           ; peripheral base at 400H, 2 ws
MPCS.value           equ 81B8H           ; PCS5 and 6 supplies,
:                                       ; peripherals in I/O space

initialize           proc far
:                                       ; determine if this is an
:                                       ; 8086 or an 80186 (checks
:                                       ; to see if the multiple bit
:                                       ; shift value was ANDed)
mov AX,2
mov CL,33
shr AX,CL
test AX,1
jz not.80186

mov DX,UMCS.reg      ; program the UMCS register
mov AX,UMCS.value
out DX,AX

mov DX,LMCS.reg      ; program the LMCS register
mov AX,LMCS.value
out DX,AX

mov DX,PACS.reg      ; set up the peripheral chip
:                                       ; selects (note the mid-range
:                                       ; memory chip selects are not
:                                       ; needed in this system, and
:                                       ; are thus not initialized)

mov AX,PACS.value
out DX,AX
mov DX,MPCS.reg
mov AX,MPCS.value
out DX,AX

:
:   Now that the chip selects are all set up, the main program of the
:   computer may be executed.
:
not.80186:
:
initialize           jmp far ptr monitor
initLhw              endp
:               ends
:               end

```

210973-94

APPENDIX G

80186 WAIT STATE PERFORMANCE

Because the 80186 contains separate bus interface and execution units, the actual performance of the processor will not degrade at a constant rate as wait states are added to the memory cycle time from the processor. The actual rate of performance degradation will depend on the type and mix of instructions actually encountered in the user's program.

Shown below are two 80186 assembly language programs, and the actual execution time for the two programs as wait states are added to the memory system of the processor. These programs show the two extremes to which wait states will or will not affect system performance as wait states are introduced.

Program 1 is very memory intensive. It performs many memory reads and writes using the more extensive memory addressing modes of the processor (which also take a greater number of bytes in the opcode for the instruction). As a result, the execution unit must constantly wait for the bus interface unit to fetch and perform the memory cycles to allow it to continue. Thus, the execution time of this type of routine will grow quickly as wait states are added, since the execution time is almost totally limited to the speed at which the processor can run bus cycles.

Note also that this program execution time calculated by merely summing up the number of clock cycles given in the data sheet will typically be less than the actual number of clock cycles actually required to run the program. This is because the numbers quoted in the data sheet assume that the opcode bytes have been prefetched and reside in the 80186 prefetch queue for immediate access by the execution unit. If the execution

unit cannot access the opcode bytes immediately upon request, dead clock cycles will be inserted in which the execution unit will remain idle, thus increasing the number of clock cycles required to complete execution of the program.

On the other hand, program 2 is more CPU intensive. It performs many integer multiplies, during which time the bus interface unit can fill up the instruction prefetch queue in parallel with the execution unit performing the multiply. In this program, the bus interface unit can perform bus operations faster than the execution unit actually requires them to be run. In this case, the performance degradation is much less as wait states are added to the memory interface. The execution time of this program is closer to the number of clock cycles calculated by adding the number of cycles per instruction because the execution unit does not have to wait for the bus interface unit to place an opcode byte in the prefetch queue as often. Thus, fewer clock cycles are wasted by the execution unit laying idle for want of instructions. Table G-1 lists the execution times measured for these two programs as wait states were introduced with the 80186 running at 8 MHz.

Table G-1

# of Wait States	Program 1		Program 2	
	Exec Time (μsec)	Perf Degr	Exec Time (μsec)	Perf Degr
0	505		294	
1	595	18%	311	6%
2	669	12%	337	8%
3	752	12%	347	3%

```

Smod186
name
;
; This file contains two programs which demonstrate the 80186 performance
; degradation as wait states are inserted. Program 1 performs a
; transformation between two types of characters sets, then copies
; the transformed characters back to the original buffer (which is 64
; bytes long. Program 2 performs the same type of transformation, however
; instead of performing a table lookup, it multiplies each number in the
; original 32 word buffer by a constant (3, note the use of the integer
; immediate multiply instruction). Program "nothing" is used to measure
; the call and return times from the driver program only.
;
cgroup      group   code
dgroup      group   data
data        segment

public 'data'

```

210973-95

push	SI	save registers used
push	CX	
push	BX	
push	AX	
mov	CX,64	translate 64 bytes
mov	SI,0	
mov	DI,0	

```

loop.back:
    mov     BL, Lstring[SI]           ; get the byte
    AL, Table[BX]                   ; translate byte
    mov     Lstring[SI], AL          ; and store it
    inc     SI                       ; increment index
    loop    loop.back               ; do the next byte

    pop     AX                      ;
    pop     BX                      ;
    pop     CX                      ;
    pop     SI                      ;
    ret                             ;

bench.1:
    endp

    proc    near
    push    AX                      ; save registers used
    push    SI
    push    CX

    mov     CX, 32                  ; multiply 32 numbers
    mov     SI, offset marray

```

Program 1		Program 2	
Exec Time (usec)	Pct Degt	Exec Time (usec)	Pct Degt
508		504	
505	18%	311	6%
509	12%	337	6%
752	15%	347	3%

5-74

```

nothing.    proc    near
            ret
nothing.    endp

; wait_state(n) sets the 80186 LMCS register to the number of wait states
; (0 to 3) indicated by the parameter n (which is passed on the stack).
; No other bits of the LMCS register are modified.
;
wait_state. proc    near
            enter    0,0           ; set up stack frame
            push     AX            ; save registers used
            push     BX
            push     DX

            mov     BX,word ptr [BP + 4] ; get argument
            mov     DX,0FFFA2h         ; get current LMCS register

            in      AX,DX

            and     AX,0FFFCb         ; and off existing ready bits
            and     BX,3              ; insure ws count is good
            or      AX,BX             ; adjust the ready bits
            out     DX,AX             ; and write to LMCS

```

```

            pop     DX
            pop     BX
            pop     AX
            leave    ; tear down stack frame
            ret
wait_state. endp

; set_timer() initializes the 80186 timers to count microseconds. Timer 2
; is set up as a prescaler to timer 0, the microsecond count can be read
; directly out of the timer 0 count register at location FF50H in I/O
; space.
;
set_timer.  proc    near
            push     AX
            push     DX

            mov     DX,0ff66h         ; stop timer 2
            mov     AX,4000h
            out     DX,AX

            mov     DX,0ff50h         ; clear timer 0 count
            mov     AX,0
            out     DX,AX

            mov     DX,0ff52h         ; timer 0 counts up to 65535
            mov     AX,0
            out     DX,AX

```

210973-97

APPENDIX H

80186 NEW INSTRUCTIONS

The 80186 performs many additional instructions to those of the 8086. These instructions appear shaded in the instruction set summary at the back of the 80186 data sheet. This appendix explains the operation of these new instructions. In order to use these new instructions with the 8086/186 assembler, the "\$mod186" switch must be given to the assembler. This can be done by placing the line: "\$mod186" at the beginning of the assembly language file.

PUSH IMMEDIATE

This instruction allows immediate data to be pushed onto the processor stack. The data can be either an immediate byte or an immediate word. If the data is a byte, it will be sign extended to a word before it is pushed onto the stack (since all stack operations are word operations).

PUSHA, POPA

These instructions allow all of the general purpose 80186 registers to be saved on the stack, or restored from the stack. The registers saved by this instruction (in the order they are pushed onto the stack) are AX, CX, DX, BX, SP, BP, SI, and DI. The SP value pushed onto the stack is the value of the register before the first PUSH (AX) is performed; the value popped for the SP register is ignored.

This instruction does not save any of the segment registers (CS, DC, SS, ES), the instruction pointer (IP), the flag register, or any of the integrated peripheral registers.

IMUL BY AN IMMEDIATE VALUE

This instruction allows a value to be multiplied by an immediate value. The result of this operation is 16 bits long. One operand for this instruction is obtained using one of the 80186 addressing modes (meaning it can be in a register or in memory). The immediate value can be either a byte or a word, but will be sign extended if it is a byte. The 16-bit result of the multiplication can be placed in any of the 80186 general purpose or pointer registers.

This instruction requires three operands: the register in which the result is to be placed, the immediate value,

and the second operand. Again, this second operand can be any of the 80186 general purpose registers or a specified memory location.

SHIFTS/ROTATES BY AN IMMEDIATE VALUE

The 80186 can perform multiple bit shifts or rotates where the number of bits to be shifted is specified by an immediate value. This is different from the 8086, where only a single bit shift can be performed, or a multiple shift can be performed where the number of bits to be shifted is specified in the CL register.

All of the shift/rotate instructions of the 80186 allow the number of bits shifted to be specified by an immediate value. Like all multiple bit shift operations performed by the 80186, the number of bits shifted is the number of bits specified modulus 32 (i.e., the maximum number of bits shifted by the 80186 multiple bit shifts is 31).

These instructions require two operands: the operand to be shifted (which may be a register or a memory location specified by any of the 80186 addressing modes) and the number of bits to be shifted.

BLOCK INPUT/OUTPUT

The 80186 adds two new input/output instructions: INS and OUTS. These instructions perform block input or output operations. They operate similarly to the string move instructions of the processor.

The INS instruction performs block input from an I/O port to memory. The I/O address is specified by the DX register; the memory location is pointed to by the DI register. After the operation is performed, the DI register is adjusted by 1 (if a byte input is specified) or by 2 (if a word input is specified). The adjustment is either an increment or a decrement, as determined by the Direction bit in the flag register of the processor. The ES segment register is used for memory addressing, and cannot be overridden. When preceded by a REPEAT prefix, this instruction allows blocks of data to be moved from an I/O address to a block of memory. Note that the I/O address in the DX register is not modified by this operation.

The OUTS instruction performs block output from memory to an I/O port. The I/O address is specified by the DX register; the memory location is pointed to by the SI register. After the operation is performed, the SI register is adjusted by 1 (if a byte output is specified) or by 2 (if a word output is specified). The adjustment is either an increment or a decrement, as determined by the Direction bit in the flag register of the processor. The DS segment register is used for memory addressing, but can be overridden by using a segment override prefix. When preceded by a REPEAT prefix, this instruction allows blocks of data to be moved from a block of memory to an I/O address. Again note that the I/O address in the DX register is not modified by this operation.

Like the string move instruction, these two instructions require two operands to specify whether word or byte operations are to take place. Additionally, this determination can be supplied by the mnemonic itself by adding a "B" or "W" to the basic mnemonic, for example:

INSB ;perform byte input

REP OUTSW ;perform word block output

BOUND

The 80186 supplies a BOUND instruction to facilitate bound checking of arrays. In this instruction, the calculated index into the array is placed in one of the general

```
PUSH BP ;save the previous frame
pointer*/
if level=0 then
    BP:=SP;
else templ:=SP; /*save current frame pointer
*/
temp2:= level - 1;
do while temp2>0 /*copy down previous level
frame*/
    BP:= BP - 2; /*pointers*/
    PUSH [BP];
    BP:=templ;
    PUSH BP; /*put current level frame
pointer*/
/*in the save area*/
SP:=SP - disp; /*create space on the stack
for*/
/*local variables*/
```

purpose registers of the 80186. Located in two adjacent word memory locations are the lower and upper bounds for the array index. The BOUND instruction compares the register contents to the memory locations, and if the value in the register is not between the values in the memory locations, an interrupt type 5 is generated. The comparisons performed are SIGNED comparisons. A register value equal to either the upper bound or the lower bound will not cause an interrupt.

This instruction requires two arguments: the register in which the calculated array index is placed, and the word memory location which contains the lower bound of the array (which can be specified by any of the 80186 memory addressing modes). The memory location containing the upper bound of the array must follow immediately the memory location containing the lower bound of the array.

ENTER AND LEAVE

The 80186 contains two instructions which are used to build and tear down stack frames of higher level, block structured languages. The instruction used to build these stack frames is the ENTER instruction. The algorithm for this instruction is:

Figure H-1 shows the layout of the stack before and after this operation.

This instruction requires two operands: the first value (disp) specifies the number of bytes the local variables of this routine require. This is an unsigned value and can be as large as 65535. The second value (level) is an unsigned value which specifies the level of the procedure. It can be as great as 255.

The 80186 includes the LEAVE instruction to tear down stack frames built up by the ENTER instruction.

As can be seen from the layout of the stack left by the ENTER instruction, this involves only moving the contents of the BP register to the SP register, and popping the old BP value from the stack.

Neither the ENTER nor the LEAVE instructions save any of the 80186 general purpose registers. If they must be saved, this must be done in addition to the ENTER and the LEAVE. In addition, the LEAVE instruction does not perform a return from a subroutine. If this is desired, the LEAVE instruction must be explicitly followed by the RET instruction.

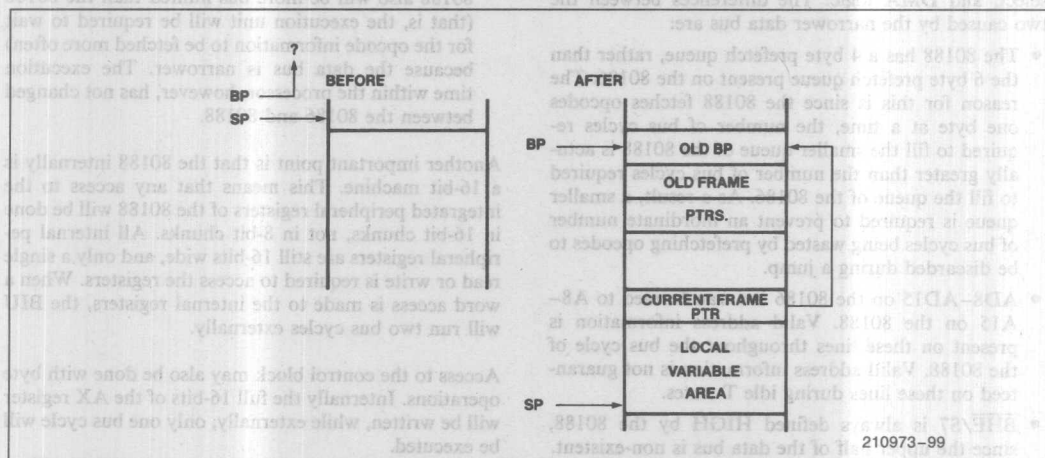


Figure H-1. ENTER Instruction Stack Frame

APPENDIX I

80186/80188 DIFFERENCES

The 80188 is exactly like the 80186, except it has an 8 bit external bus. It shares the same execution unit, timers, peripheral control block, interrupt controller, chip select, and DMA logic. The differences between the two caused by the narrower data bus are:

- The 80188 has a 4 byte prefetch queue, rather than the 6 byte prefetch queue present on the 80186. The reason for this is since the 80188 fetches opcodes one byte at a time, the number of bus cycles required to fill the smaller queue of the 80188 is actually greater than the number of bus cycles required to fill the queue of the 80186. As a result, a smaller queue is required to prevent an inordinate number of bus cycles being wasted by prefetching opcodes to be discarded during a jump.
- AD8-AD15 on the 80186 are transformed to A8-A15 on the 80188. Valid address information is present on these lines throughout the bus cycle of the 80188. Valid address information is not guaranteed on these lines during idle T states.
- $\overline{\text{BHE}}/\text{S7}$ is always defined HIGH by the 80188, since the upper half of the data bus is non-existent.
- The DMA controller of the 80188 only performs byte transfers. The $\overline{\text{B}}/\text{W}$ bit in the DMA control word is ignored.

- Execution times for many memory access instructions are increased because the memory access must be funnelled through a narrower data bus. The 80188 also will be more bus limited than the 80186 (that is, the execution unit will be required to wait for the opcode information to be fetched more often) because the data bus is narrower. The execution time within the processor, however, has not changed between the 80186 and 80188.

Another important point is that the 80188 internally is a 16-bit machine. This means that any access to the integrated peripheral registers of the 80188 will be done in 16-bit chunks, not in 8-bit chunks. All internal peripheral registers are still 16-bits wide, and only a single read or write is required to access the registers. When a word access is made to the internal registers, the BIU will run two bus cycles externally.

Access to the control block may also be done with byte operations. Internally the full 16-bits of the AX register will be written, while externally, only one bus cycle will be executed.

APPENDIX J

80186/80C186 DIFFERENCES

There are two operating modes of the 80C186 and 80C188: Compatible Mode and Enhanced Mode. In Compatible Mode, the 80C186 will function identically to the 80186 with the following noted exceptions:

- 1) All non-initialized registers in the peripheral control block will reset to a random value on power-up on the 80C186. Non-Initialized registers consist of those registers which are not used for control, i.e., address pointers, max count, etc. For compatibility, all registers should be programmed before being used on existing 80186 applications as well as on new 80C186 applications.
- 2) The ET (Esc/Trap) bit in the relocation register has no effect in Compatible Mode. If an escape opcode is executed, the 80C186 will always trap to an interrupt vector type 7. The 80C186 does not support any numerics operations when in Compatible Mode.

In Enhanced Mode, the 80C186 provides additional features not found on the 80186. There are newly defined registers to support these new features, and three of the output pins of the 80C186 change functionality. The new registers and pin descriptions are covered in Section 9.0.

The 80C188 in Enhanced Mode functions similarly to the 80C186 except for numerics operation. It is not possible to interface a numerics coprocessor with the 80C188. Therefore, none of the MCS pins change functionality when invoking Enhanced Mode on the 80C188. Further, any attempted execution of an escape opcode will result in a trap to interrupt vector type 7.

NOTE: Address bit A0 can be used in either RAS or CAS addresses, so long as it is not included in any refresh address bits.

Column Address (A0-A7)	Row Address (A8-A15)		
A8-A15	A1-A7, A0	(84K Bytes)	84K x 1
A8-A15	A1-A7, A0	(16K Bytes)	16K x 4
A8-A15	A1-A7, A0	(328K Bytes)	328K x 1
A0, A8-A15	A1-A7	(84K Bytes)	84K x 4
A10-A15	A1-A7, A0	(1M Bytes)	1M x 1
A0, A10-A15	A1-A7	(328K Bytes)	328K x 4

Refresh Add	CAS Add	RAS Add	RAM Type
A0-A7	A0-A7	A0-A7	84K x 1
A0-A7	A0-A7	A0-A7	16K x 4
A0-A7	A0-A7	A0-A7	328K x 1
A0-A7	A0-A7	A0-A7	84K x 4
A0-A7	A0-A7	A0-A7	1M x 1
A0-A7	A0-A7	A0-A7	328K x 4

APPENDIX K

DRAM ADDRESSING CONFIGURATIONS FOR THE 80C186/80C188

80C186 DESIGNS

		Row Address (A0-AX)	Column Address (A0-AX)
64K x 1	(128K Bytes)	A1-A8	A9-A16
16K x 4	(32K Bytes)	A1-A8	A9-A14
256K x 1	(512K Bytes)	A1-A9	A10-A18
64K x 4	(128K Bytes)	A1-A8	A9-A16
1M x 1	(2M Bytes)	A1-A10	A11-A19 (+ Bank)
256K x 4	(512K Bytes)	A1-A9	A10-A18

80C188 DESIGNS

NOTE:

Address bit A0 can be used in either $\overline{\text{RAS}}$ or $\overline{\text{CAS}}$ addresses, so long as it is not included in any refresh address bits.

		Row Address (A0-AX)	Column Address (A0-AX)
64K x 1	(64K Bytes)	A1-A7, A0	A8-A15
16K x 4	(16K Bytes)	A1-A7, A0	A8-A13
256K x 1	(256K Bytes)	A1-A8, A0	A9-A17
64K x 4	(64K Bytes)	A1-A8	A0, A9-A15
1M x 1	(1M Bytes)	A1-A9, A0	A10-A19
256K x 4	(256K Bytes)	A1-A9	A0, A10-A17

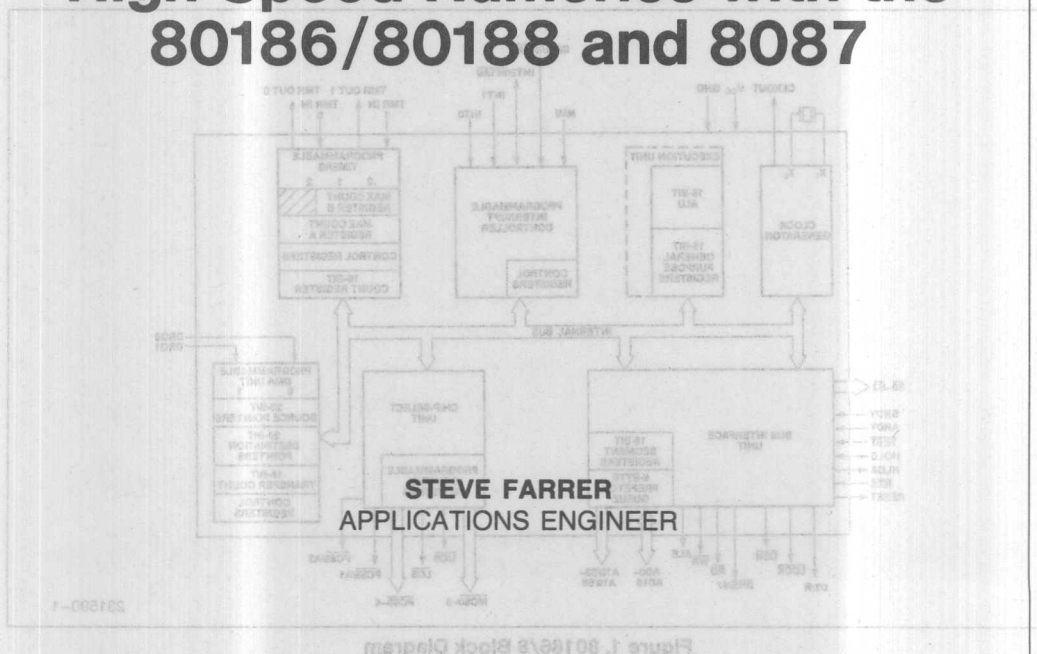
RAM Type	$\overline{\text{RAS}}$ Add	$\overline{\text{CAS}}$ Add	Refresh Add
64K x 1	A0-A7	A0-A7	A0-A6
16K x 4	A0-A7	A0-A5	A0-A6
256K x 1	A0-A8	A0-A8	A0-A7
64K x 4	A0-A7	A0-A7	A0-A7
1M x 1	A0-A9	A0-A9	A0-A8
256K x 4	A0-A8	A0-A8	A0-A8

February 1986

meric with the

High Speed Numerics with the 80186/80188 and 8087

STEVE FARRER
APPLICATIONS ENGINEER



1.0 INTRODUCTION

From their introduction in 1982, the highly integrated 16-bit 80186 and its 8-bit external bus version, the 80188, have been ideal processor choices for high-performance, low-cost embedded control applications. The integrated peripheral functions and enhanced 8086 CPU of the 80186 and 80188 allow for an easy upgrade of older generation control applications to achieve higher performance while lowering the overall system cost through reduced board space, and a simplified production flow.

More and more controller applications need even higher performance in numerics, yet still require the low-cost and small form factor of the 80186 and 80188. The 8087 Numerics Data Coprocessor satisfies this need as an optional add-on component.

The 8087 Numeric Data Coprocessor is interfaced to the 80186 and 80188 through the 82188 IBC (Integrated Bus Controller). The IBC provides a highly integrated interface solution which replaces the 8288 used in 8086-8087 systems. The IBC incorporates all the necessary bus control for the 8087 while also providing the necessary logic to support the interface between the 80186/8 and the 8087.

This application note discusses the design considerations associated with using the 8087 Numeric Data Coprocessor with the 80186 and 80188. Sections two,

three, and four contain an overview of the integrated circuits involved in the numerics configuration. Section five discusses the interfacing aspects between the 80186/8 and the 8087, including the role of the 82188 Integrated Bus Controller and the operation of the integrated peripherals on the 80186/8 with the 8087. Section six compares the advantages of using an 8087 Numeric Data Coprocessor over software routines written for the host processor as well as the advantage of using an 80186/8 numerics system over an 8086/8088 numerics system.

Except where noted, all future references to the 80186 will apply equally to the 80188.

2.0 OVERVIEW OF THE 80186

The 80186 and 80188 are highly integrated microprocessors which effectively combine up to 20 of the most common system components onto a single chip. The 80186 and 80188 processors are designed to provide both higher performance and a more highly integrated solution to the total system.

Higher integration results from integrating system peripherals onto the microprocessor. The peripherals consist of a clock generator, an interrupt controller, a DMA controller, a counter/timer unit, a programmable wait state generator, programmable chip selects, and a bus controller. (See Figure 1.)

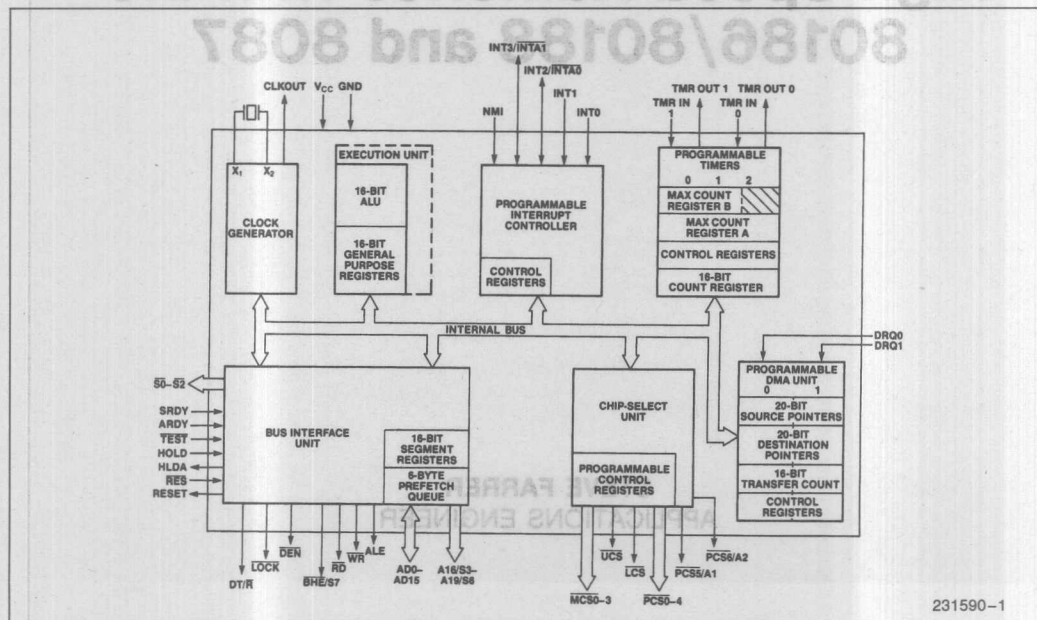


Figure 1. 80186/8 Block Diagram

Higher performance results from enhancements to both general and specific areas of the 8086 CPU, including faster effective address calculation, improvement in the execution speed of many instructions, and the inclusion of new instructions which are designed to produce optimum 80186 code.

The 80186 and 80188 are completely object code compatible with the 8086 and 8088. They have the same basic register set, memory organization, and addressing modes. The differences between the 80186 and 80188 are the same as the differences between the 8086 and 8088: the 80186 has a 16-bit architecture and 16-bit bus interface; the 80188 has a 16-bit internal architecture and an 8-bit data bus interface. The instruction execution times of the two processors differ accordingly: for each non-immediate 16-bit data read/write instruction, 4 additional clock cycles are required by the 80188.

3.0 NUMERICS OVERVIEW

3.1 The Benefits of Numeric Coprocessing

The 8086/8 and 80186/8 are general purpose microprocessors, designed for a very wide range of applications. Typically, these applications need fast, efficient data movement and general purpose control instructions. Arithmetic on data values tends to be simple in these applications. The 8086/8 and 80186/8 fulfill these needs in a low cost, effective manner.

However, some applications require extremely fast and complex math functions which are not provided by a general purpose processor. Such functions as square root, sine, cosine, and logarithms are not directly available in a general purpose processor. Software routines required to implement these functions tend to be slow and not very accurate. Integer data types and their arithmetic operations (i.e., add, subtract, multiply and divide) which are directly available on general purpose processors, still may not meet the needs for accuracy, speed and ease of use.

Providing fast, accurate, complex math can be quite complicated, requiring large areas of silicon on integrated circuits. A general data processor does not provide these features due to the extra cost burden that less complex general applications must take on. For such features, a special numeric data processor is required — one which is easy to use and has a high level of support in hardware and software.

3.2 Introduction to the 8087

The 8087 is a numeric data coprocessor which is capable of performing complex mathematical functions while the host processor (i.e. the main CPU) performs

more general tasks. It supports the necessary data types and operations and allows use of all the current hardware and software support for the 8086/8 and 80186/8 microprocessors. The fact that the 8087 is a coprocessor means it is capable of operating in parallel with the host CPU, which greatly improves the processing power of the system.

The 8087 can increase the performance of floating-point calculations by 50 to 100 times, providing the performance and precision required for small business and graphics applications as well as scientific data processing.

The 8087 numeric coprocessor adds 68 floating-point instructions and eight 80-bit floating-point registers to the basic 8086 programming architecture. All the numeric instructions and data types of the 8087 are used by the programmer in the same manner as the general data types and instructions of the host.

The numeric data formats and arithmetic operations provided by the 8087 support the proposed IEEE Microprocessor Floating Point Standard. All of the proposed IEEE floating point standard algorithms, exception detection, exception handling, infinity arithmetic and rounding controls are implemented. The IEEE standard makes it easier to use floating point and helps to avoid common problems that are inherent to floating point.

3.3 Escape Instructions

The coprocessing capabilities of the 8087 are achieved by monitoring the local bus of the host processor. Certain instructions within the 8086 assembly language known as ESCAPE instructions are defined to be coprocessor instructions and, as such, are treated differently.

The coprocessor monitors program execution of the host processor to detect the occurrence of an ESCAPE instruction. The fetching of instructions is monitored via the data bus and bus cycle status S2-S0, while the execution of instructions is monitored via the queue status lines QS0 and QS1.

All ESCAPE instructions start with the high-order 5-bits of the instruction opcode being 11011. They have two basic forms, the memory reference form and the non-memory reference form. The non-memory form, shown in Figure 2A, initiates some activity in the coprocessor using the nine available bits of the ESCAPE instruction to indicate which function to perform.

Memory reference forms of the ESCAPE instruction, shown in Figure 2B, allow the host to point out a memory operand to the coprocessor using any host memory

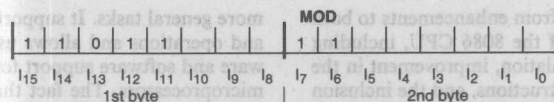


Figure 2A. Non-Memory Reference ESCAPE Instructions

addressing mode. Six bits are available in the memory reference form to identify what to do with the memory operand.

Memory reference forms of ESCAPE instructions are identified by bits 7 and 6 of the byte following the ESCAPE opcode. These two bits are the MOD field of the 8086/8 or 80186/8 effective address calculation byte. Together with the R/M field (bits 2 through 0), they determine the addressing mode and how many subsequent bytes remain in the instruction.

3.4 Host Response to Escape Instructions

The host performs one of two possible actions when encountering an ESCAPE instruction: do nothing (operation is internal to 8087) or calculate an effective address and read a word value beginning at that address (required for all LOADS and STORES). The host ignores the value of the word read and hence the cycle is referred to as a "Dummy Read Cycle." ESCAPE instructions do not change any registers in the host other than advancing the IP. If there is no coprocessor or the coprocessor ignores the ESCAPE instruction, the ESCAPE instruction is effectively a NOP to the host. Other than calculating a memory address and reading a word of memory, the host makes no other assumptions regarding coprocessor activity.

The memory reference ESCAPE instructions have two purposes: to identify a memory operand and, for certain instructions, to transfer a word from memory to the coprocessor.

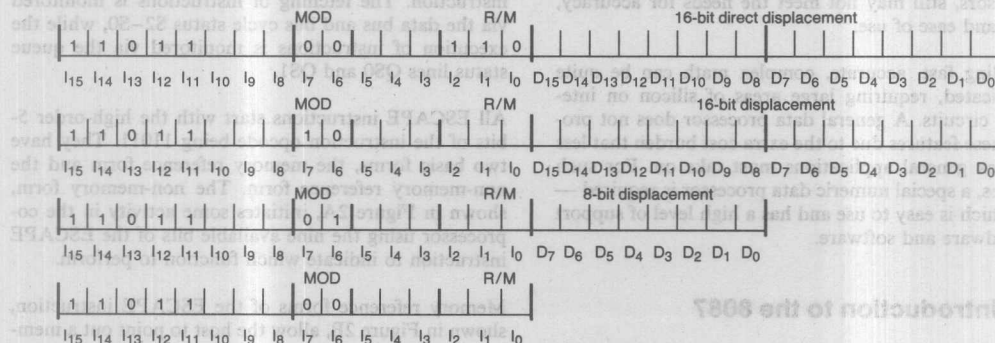


Figure 2B. Memory Reference ESCAPE Instruction Forms

3.5 Coprocessor Response to Escape Instructions

The 8087 performs basically three types of functions when encountering an ESCAPE instruction: LOAD (read from memory), STORE (write to memory), and EXECUTE (perform one of the internal 8087 math functions).

When the host executes a memory reference ESCAPE instruction intended to cause a read operation by the 8087, the host always reads the low-order word of any 8087 memory operand. The 8087 will save the address and data read. To read any subsequent words of the operand, the 8087 must become a local bus master.

When the 8087 has the local bus, it increments the 20-bit physical address it saved to address the remaining words of the operand.

When the ESCAPE instruction is intended to cause a write operation by the 8087, the 8087 will save the address but ignore the data read. Eventually, it will get control of the local bus and perform successive writes incrementing the 20-bit address after each word until the entire numeric variable has been written.

ESCAPE instructions intended to cause the execution of a coprocessor calculation do not require any bus activity. Numeric calculations work off of an internal register stack which has been initialized using a LOAD operation. The calculation takes place using one or two of the stack positions specified by the ESCAPE instruction. The result of the operation is also placed in one of the stack positions specified by the ESCAPE instruction. The result may then be returned to memory using a STORE instruction, thus allowing the host processor to access it.

4.0 OVERVIEW OF THE 82188 INTEGRATED BUS CONTROLLER

4.1 Introduction

The 82188 Integrated Bus Controller (IBC) is a highly integrated version of the 8288 Bus Controller. The IBC provides command and control timing signals for bus control and all of the necessary logic to interface the 80186 to the 8087.

4.2 Bus Control Signals

The bus command and control signals consist of \overline{RD} , \overline{WR} , \overline{DEN} , $\overline{DT/R}$, and \overline{ALE} . The timings and levels are driven following the latching of valid signals on the status lines $S0-S2$. When $S0-S2$ change state from passive to active, the IBC begins cycling through a state machine which drives the corresponding control and command lines for the bus cycle. As with the 8288, an address enable input (\overline{AEN}) is present to allow tri-stat-

ing when other bus masters supply their own bus control signals.

4.3 Bus Arbitration

The IBC also has the ability to convert bus arbitration protocols of $\overline{RQ}/\overline{GT}$ to \overline{HOLD} - \overline{HLDA} . This allows the 82586 Local Area Network (LAN) Coprocessor, the 82730 Text Coprocessor, and other coprocessors using the \overline{HOLD} - \overline{HLDA} protocol to be interfaced to the 8086/8 as well as allowing the 80186/8 to be interfaced to the 8087. In addition to converting arbitration protocols, the IBC makes it possible to arbitrate between two bus masters using \overline{HOLD} - \overline{HLDA} with a third using $\overline{RQ}/\overline{GT}$.

4.4 Interface Logic

In addition to all the bus control and arbitration features, the IBC provides logic to connect the queue status to the 8087, a chip-select for the 8087, and the necessary \overline{READY} synchronization required between the 8087 and the 80186/8.

5.0 DESIGNING THE SYSTEM

5.1 Circuit Schematics of the 80186/8-82188-8087 System

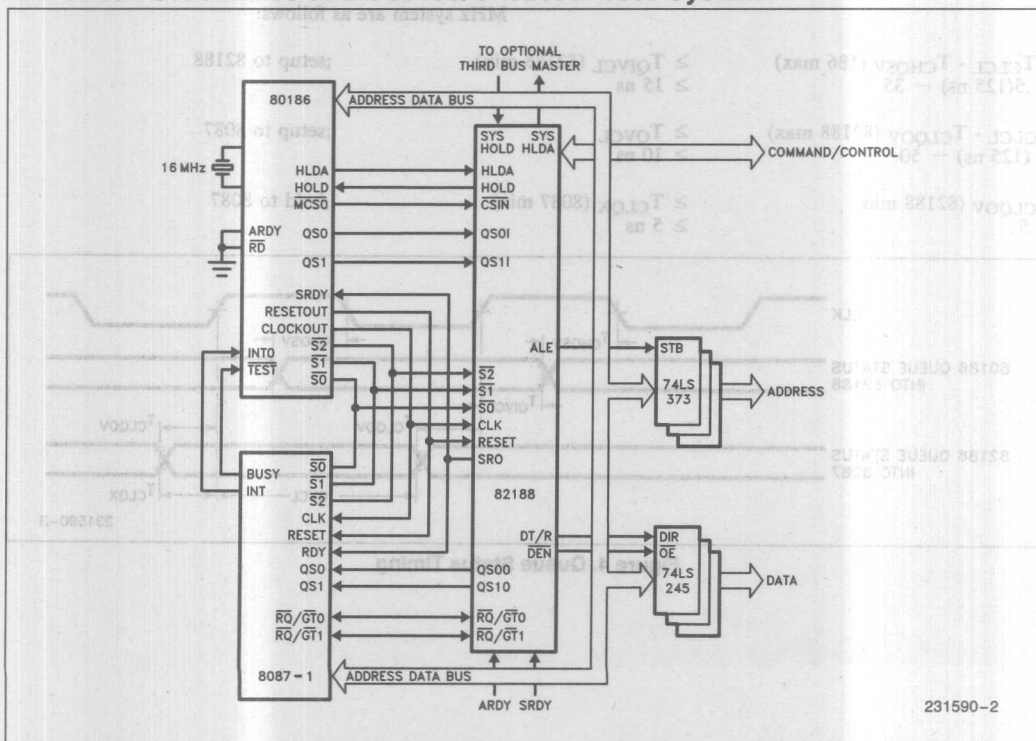


Figure 3. 80186/8-82188-8087 Circuit Diagram

5.2 Queue Status

The 8087 tracks the instruction execution of the 80186 by keeping an internal instruction queue which is identical to the processor's instruction queue. Each time the processor performs an instruction fetch, the 8087 latches the instruction into its own queue in parallel with the processor. Each time the processor removes the first byte of an instruction from the queue, the 8087 removes the byte at the top of the 8087 queue and checks to see if the byte is an ESCAPE prefix. If it is, the 8087 decodes the following bytes in parallel with the processor to determine which numeric instruction the bytes represent. If the first byte of the instruction is not an ESCAPE prefix, the 8087 discards it along with the subsequent bytes of the non-numeric instruction as the 80186 removes them from the queue for execution.

The 8087 operates its internal instruction queue by monitoring the two queue status lines from the CPU. This status information is made available by the CPU by placing it into queue status mode. This requires strapping the \overline{RD} pin on the 80186 to ground. When \overline{RD} is tied to ground, ALE and \overline{WR} become QS0 (Queue Status #0) and QS1 (Queue Status #1) respectively.

Table 1. Queue Status Decoding

QS1	QS0	Queue Operation
0	0	No queue operation
0	1	First byte from queue
1	0	Subsequent byte from queue
1	1	Reserved

Each time the 80186 begins decoding a new instruction, the queue status lines indicate "first byte of instruction taken from the queue". This signals the 8087 to check for an ESCAPE prefix. As the remaining bytes of the instruction are removed, the queue status indicates "subsequent byte removed from queue". The 8087 uses this status to either continue decoding subsequent bytes, if the first byte was an ESCAPE prefix, or to discard the subsequent bytes if the first byte was not an ESCAPE prefix.

The QS0(ALE) and QS1(\overline{WR}) pins of the 80186 are fed directly to the 82188 where they are latched and delayed by one-half-clock. The delayed queue status from the 82188 is then presented directly to the 8087.

The waveforms of the queue status signals are shown in Figure 4. The critical timings are the setup time into the 82188 from the 80186 and the setup and hold time into the 8087 from the 82188. The calculations for an 8 MHz system are as follows:

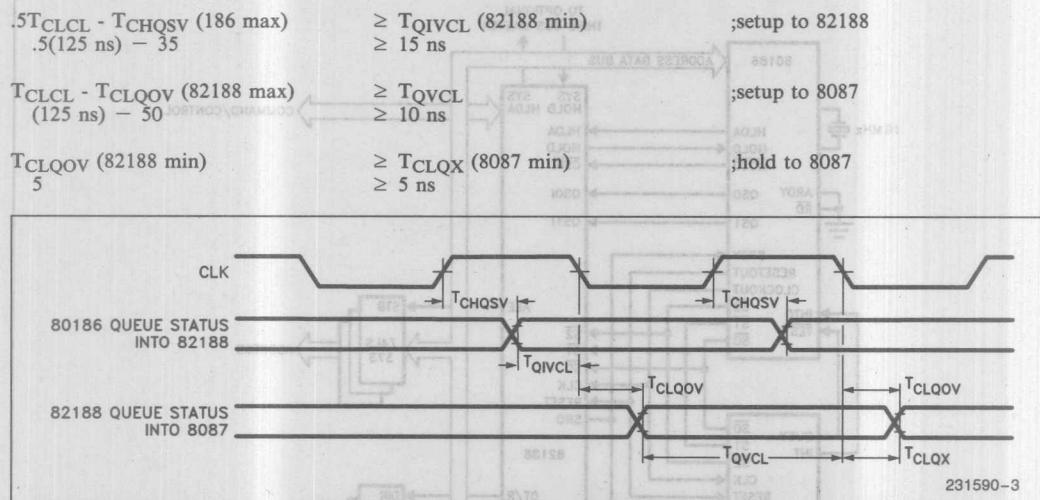


Figure 4. Queue Status Timing

5.3 Bus Control Signals

When the 80186 is in Queue Status mode, another component must generate the ALE, \overline{RD} , and \overline{WR} signals. The 82188 provides these signals by monitoring the CPU bus cycle status ($\overline{S0}$ – $\overline{S2}$). Also provided are \overline{DEN} and $\overline{DT/R}$ which may be used for extra drive capability on the control bus. With the exception of ALE, all control signals on the 82188 are almost identical to their corresponding 80186 control signals. This section discusses the differences between the 80186 and the 82188 control signals for the purpose of upgrading an 80186 design to an 80186–8087 design. For original 80186–8087 designs, there is no need to compare control signal timings of the 82188 with the 80186.

5.3.1 ALE

The ALE (Address Latch Enable) signal goes active one clock phase earlier on the 80186 than on the 82188. Timing of the ALE signal on the 82188 is closer to that of the 8086 and 8288 bus controller because the bus cycle status is used to generate the ALE pulse. ALE on the 80186 goes active before the bus cycle status lines are valid.

The inactive-edge of ALE occurs in the same clock phase for both the 80186 and the 82188. The setup and hold times of the 80186 address relative to the 82188 ALE signal are shown in Figure 5 and are calculated for an 8 MHz system as follows:

Setup Time

For 80186 = T_{AVCH} (186 min) + T_{CHLL} (82188 min)
 $= 10 + 0 = 10$ ns.

For 8087 = $0.5(T_{CLCL}) - T_{CLAV}$ (8087 max) + T_{CHLL} (82188 min)
 $= 0.5(125) - 55 + 0 = 7.5$ ns.

Hold Time

$= 0.5(T_{CLCL}) - T_{CHLL}$ (82188 max) + T_{CLAZ} (186 min)
 $= 0.5(125) - 30 + 10 = 42.5$ ns.

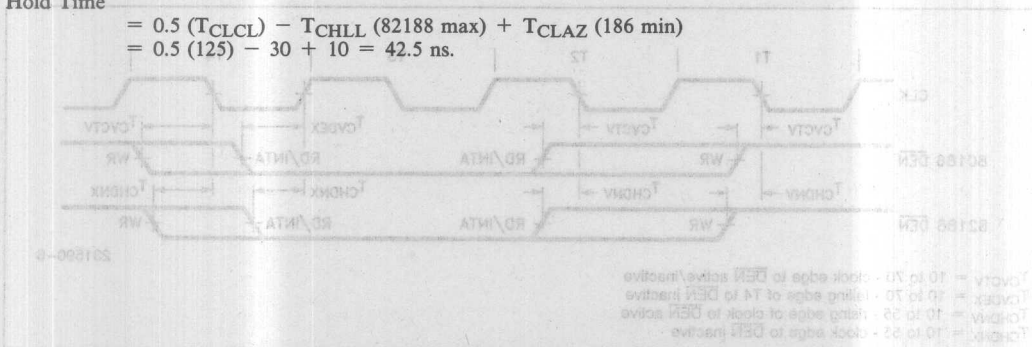


Figure 7: Data Control Timings

NOTE:

The hold time calculation is the same for both the 80186 and 8087.

These timings provide adequate setup and hold times for a 74LS373 address latch.

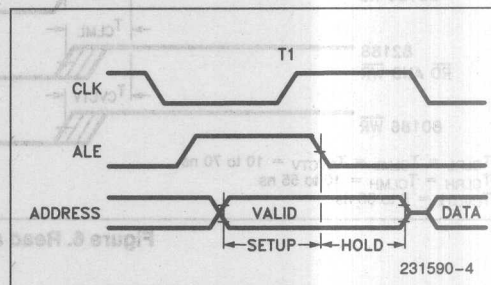


Figure 5: Address Latch Timings

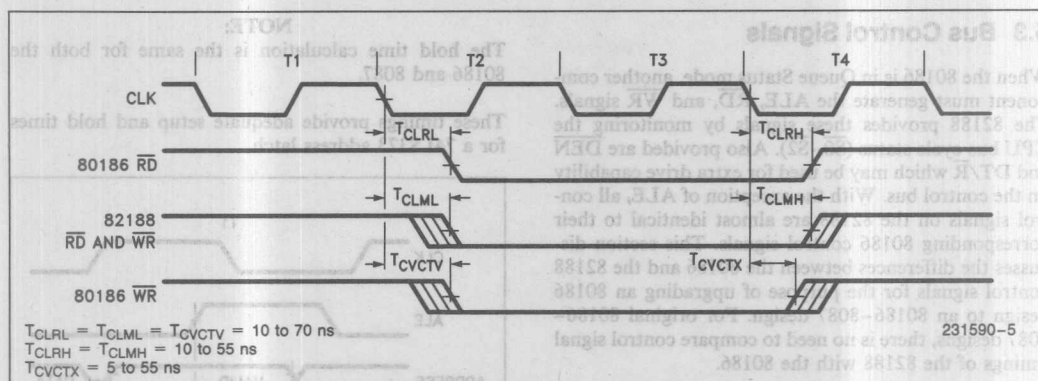


Figure 6. Read and Write Timings

5.3.2 Read and Write

The read and write signals of the 82188 have identical timings to those of the 80186 with one exception: the 82188 \overline{WR} inactive edge may not go inactive quite as early as the 80186. This spec is, in fact, a tighter spec than the 80186 \overline{WR} timing and should make designs easier. The timings for \overline{RD} and \overline{WR} are shown in Figure 6 for both the 80186 and the 82188.

5.3.3 \overline{DEN}

The \overline{DEN} signal on the 82188 is identical to the \overline{DEN} signal on the 80186 but with a tighter timing specification. This makes designs easier with the 82188 and makes upgrades from 80186 bus control to 82188 bus control more straightforward. The timings for \overline{DEN} on both the 80186 and 82188 are shown in Figure 7.

5.3.4 DT/\overline{R}

The operation of the DT/\overline{R} signal varies somewhat between the 80186 and the 82188. The 80186 DT/\overline{R} signal will remain in an active high state for all write cycles and will default to a high state when the system bus is idle (i.e., no bus activity). The 80186 DT/\overline{R} goes low only for read cycles and does so only for the duration of the bus cycle. At the end of the read cycle, assuming the following cycle is a non-read, the DT/\overline{R} signal will default back to a high state. Back-to-back read cycles will result in the DT/\overline{R} signal remaining low until the end of the last read cycle.

The DT/\overline{R} signal on the 82188 operates differently by making transitions only at the start of a bus cycle. The 82188 DT/\overline{R} signal has no default state and therefore will remain in whichever state the previous bus cycle required. The 82188 DT/\overline{R} signal will only change states when the current bus cycle requires a state different from the previous bus cycle.

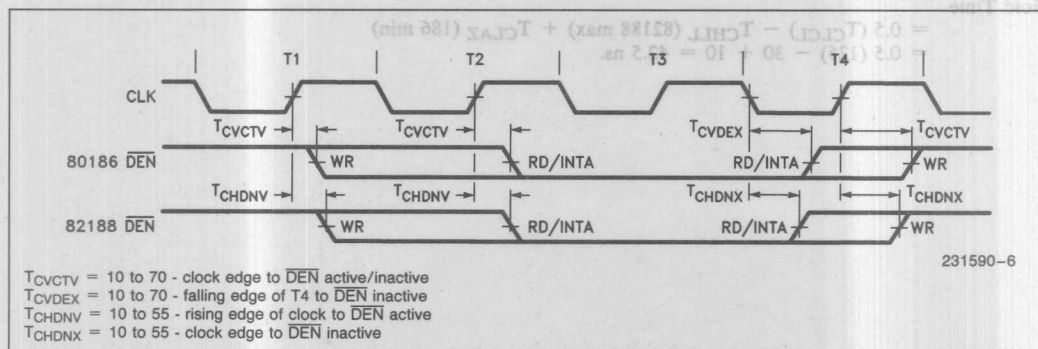


Figure 7. Data Control Timings

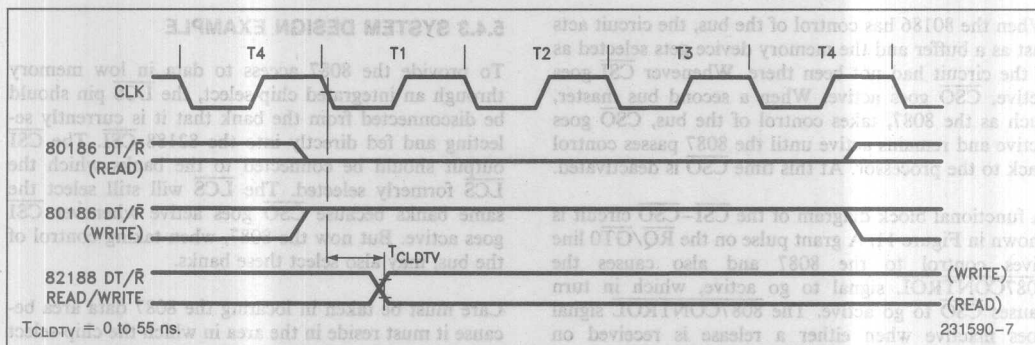


Figure 8. Data Transmit & Receive Timings

5.4 Chip Selects

5.4.1 INTRODUCTION

Chip-select circuitry is typically accomplished by using a discrete decoder to decode two or more of the upper address lines. When a valid address appears on the address bus, the decoder generates a valid chip-select. With this method, any bus master capable of placing an address on the system bus is able to generate a chip-select. An example of this is shown in Figure 9 where an 8086/8087 system uses a common decoder on the address bus. Note the decoder is able to operate regardless of which processor is in control of the bus.

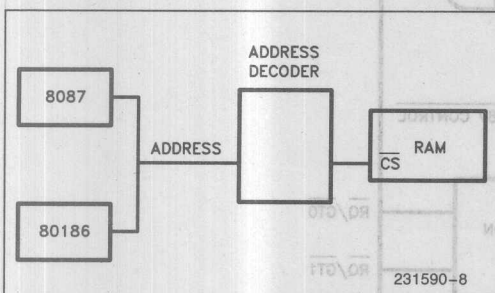


Figure 9. Typical 8086/8087 System

With high integration processors like the 80186 and 80188, the chip-select decoder is integrated onto the processor chip. The integrated chip-selects on the 80186 enable direct processor connection to the chip-enable pins on many memory devices, thus eliminating an external decoder. But because the integrated chip-selects decode the 80186's internal bus, an external bus master, such as the 8087, is unable to activate them. The 82188 IBC solves this problem by supplying a chip-select mechanism which may be activated by both the host processor and a second processor.

5.4.2 \overline{CS} AND \overline{CSO} OF THE 82188

The \overline{CSI} (chip select in) and \overline{CSO} (chip select out) pins of the 82188 provide a way for a second bus master to select memory while also making use of the 80186 integrated chip-selects. The \overline{CSI} pin of the 82188 connects directly to one of the 80186's chip-selects while \overline{CSO} connects to the memory device designated for the chip-selects range. An example of this is shown in Figure 10.

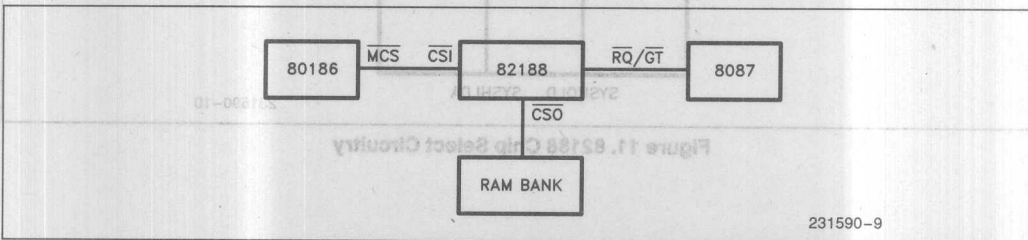


Figure 10. Typical 80186/82188/8087 System

When the 80186 has control of the bus, the circuit acts just as a buffer and the memory device gets selected as if the circuit had not been there. Whenever \overline{CS} goes active, \overline{CSO} goes active. When a second bus master, such as the 8087, takes control of the bus, \overline{CSO} goes active and remains active until the 8087 passes control back to the processor. At this time \overline{CSO} is deactivated.

A functional block diagram of the \overline{CS} - \overline{CSO} circuit is shown in Figure 11. A grant pulse on the $\overline{RQ}/\overline{GT0}$ line gives control to the 8087 and also causes the 8087CONTROL signal to go active, which in turn causes \overline{CSO} to go active. The 8087CONTROL signal goes inactive when either a release is received on $\overline{RQ}/\overline{GT0}$, indicating that the 8087 is relinquishing control to the main processor, or a grant is received on the $\overline{RQ}/\overline{GT1}$ line, indicating that the 8087 is relinquishing control to a third processor. Both actions signify that the 8087 is relinquishing the bus. If \overline{CSO} goes inactive because a third processor took control of the bus, then \overline{CSO} will go active again for the 8087 when a release pulse is transmitted on the $\overline{RQ}/\overline{GT1}$ line to the 8087. This release pulse occurs as a result of SYSHLDA going inactive from the third processor.

5.4.3 SYSTEM DESIGN EXAMPLE

To provide the 8087 access to data in low memory through an integrated chip-select, the \overline{LCS} pin should be disconnected from the bank that it is currently selecting and fed directly into the 82188 \overline{CS} . The \overline{CS} output should be connected to the banks which the \overline{LCS} formerly selected. The \overline{LCS} will still select the same banks because \overline{CSO} goes active whenever \overline{CS} goes active. But now the 8087, when taking control of the bus, may also select these banks.

Care must be taken in locating the 8087 data area because it must reside in the area in which the chip-select is defined. If the 8087 generates an address outside of the \overline{LCS} range, the \overline{CSO} will still go active, but the address will erroneously select a part of the lower bank. Note also that this chip-select limits the size of the 8087 data area to the maximum size memory which can be selected with one chip-select. However, this does not place a limit on instruction code size or non-8087 data size. All 80186 and 8087 instructions are fetched by the processor and therefore do not require that the 8087 be

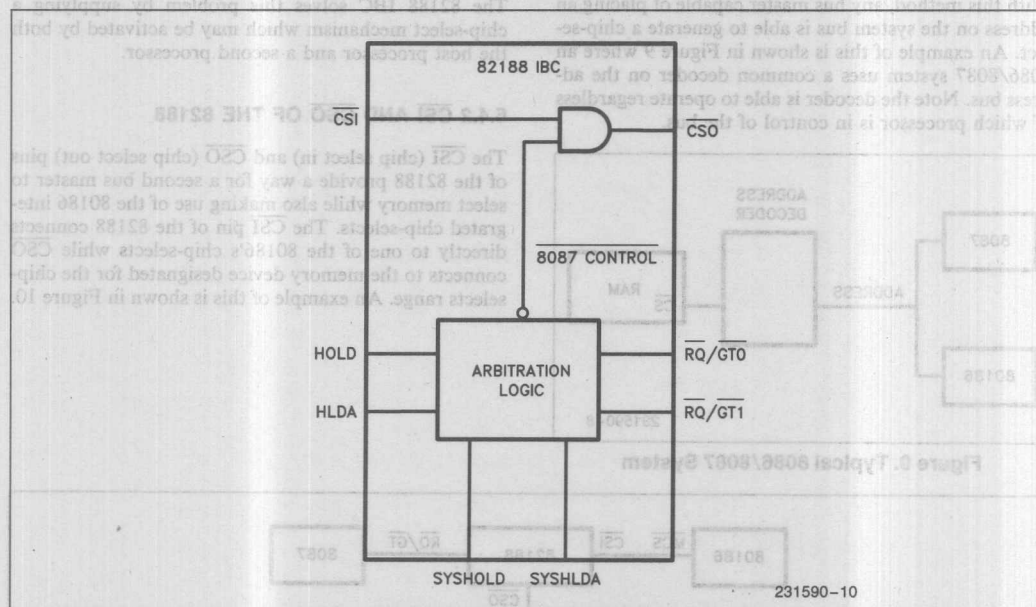


Figure 11. 82188 Chip Select Circuitry

able to address them. Likewise, non-8087 data is never accessed by the 8087 and therefore does not require an 8087 chip-select.

5.5 Wait State & Ready Logic

The 8087 must accurately track every instruction fetch the 80186 performs so that each op-code may be read from the system bus by the 8087 in parallel with the processor. This means that for instruction code areas, the 80186 cannot use internally generated wait states. All ready logic for these areas must be generated externally and sent into the 82188. The 82188 then presents a synchronous ready out (SRO) signal to both the 80186 and the 8087.

5.5.1 INTERNAL WAIT STATES WITH INSTRUCTION FETCHES

If internal wait states are used by the processor with the 8087 at zero wait states, then the 8087 will latch op-codes using a four clock bus cycle while the processor is using between five and seven clocks on each bus cycle. If the wait states are truly necessary to latch valid data from memory, then a four clock bus cycle will force the 8087 to latch invalid data. The invalid data may then be possibly interpreted to be an ESCAPE prefix when, in reality, it is not. The reverse may also hold true in that the 8087 may not recognize an ESCAPE prefix when it is fetched. These conditions could cause a system to hang (i.e., cease to operate), or operate with erroneous results.

If the memory is fast enough to allow latching of valid data within a four clock bus cycle, then the 80186 internal wait states will not cause the system to hang. Both processors will receive valid data during their respective bus cycles. The 8087 will finish its bus cycle earlier than the processor, but this is of no consequence to system operation. The 8087 will synchronize with the processor using the status lines S0-S2 at the start of the next instruction fetch.

5.5.2 INTERNAL WAIT STATES WITH DATA & I/O CYCLES

With the exception of "Dummy Read Cycles" and instruction fetches, all memory and I/O bus cycles executed by the host processor are ignored by the 8087. Coprocessor synchronization is not required for untracked bus cycles and, therefore, internally generated wait states do not affect system operation. All of the I/O space and any part of memory used strictly for data may use the internal wait state generator on the 80186.

Memory used for 8087 data is somewhat different. Here, as in the case of code segment areas, the system must rely on an external ready signal or else the memory must be fast enough to support zero wait state operation. Without an external ready signal, the 8087 will always perform a four clock bus cycle which, when used with slow memories, results in the latching of invalid data.

Internal wait states will not affect system operation for data cycles performed by the 8087. In this case the 8087 has control of the bus and the two processors operate independently.

One type of data cycle has not yet been considered. Each time a numerics variable is accessed, the host processor runs a "Dummy Read Cycle" in order to calculate the operand address for the 8087. The 8087 latches the address and then takes control of the bus to fetch any subsequent bytes which are necessary. If the 8087 variables are located at even addresses, then an internally generated wait state will not present any problems to the system. If any numeric variables are located at odd addresses, then the interface between the host and coprocessor becomes asynchronous causing erroneous results.

The erroneous results are due to the 80186 running two back-to-back bus cycles with wait states while the 8087 runs two back-to-back bus cycles without wait states. The start of the second bus cycle is completely uncoordinated between the two processors and the 8087 is unable to latch the correct address for subsequent transfers. For this reason, 8087 variables in a 80186 system must always lie on even boundaries when using the internal wait state generator to access them.

Numeric variables in an 80188 system must never be in a section of memory which uses the internal wait state generator. The 80188 will always perform consecutive bus cycles which would be equivalent to the 80186 performing an odd addressed "Dummy Read Cycle."

5.5.3 AUTOMATIC WAIT STATES AT RESET

The 80186 automatically inserts three wait states to the predefined upper memory chip select range upon power up and reset. This enables designers to use slow memories for system boot ROM if so desired. If slow ROM's are chosen, then no further programming is necessary. If fast ROM's are chosen, then the wait state logic may simply be reprogrammed to the appropriate number of wait states.

The automatic wait states have the possibility of presenting the same problem as described in section 5.5.1 if

the boot ROM needs one or more wait states. Under these conditions the 8087 would be forced to latch invalid opcodes and possibly mistake one for an ESCAPE instruction.

If the boot ROM requires wait states, then some sort of external ready logic is necessary. This allows both processors to run with the same number of wait states and insures that they always receive valid data.

If the boot ROM does not require wait states, then there is no need to design external ready logic for the upper chip select region. But if 8087 code is present in the upper memory chip select region, the situation described in section 3.4 regarding "Dummy Read Cycles" must be considered.

The 82188 solves this problem by inserting three wait states on the SRO line to the 8087 for the first 256 bus cycles. By doing this the 82188 inserts the same number of wait states to both processors keeping them synchronized. The initialization code for the 80186 must program the upper memory chip select to look at external ready and to insert zero wait states within these first 256 bus cycles. At the end of the 256 bus cycles, the 82188 stops inserting wait states and both processors run at zero wait states.

5.5.4 EXTERNAL READY SYNCHRONIZATION

The 80186 and 8087 sample READY on different clock edges. This implies that some sort of external synchronization is required to insure that both processors sample the same ready state. Without the synchronization, it would be possible for the external signal to change state between samples. The 80186 may sample ready high while the 8087 samples ready low. This would lead to the two processors running different length bus cycles and possibly cause the system to hang.

The 82188 provides ready synchronization through the ARDY and SRDY inputs. Once a valid transition is recorded, the 82188 presents the results on the SRO output and holds the output in that state until both processors have had a chance to sample the signal.

5.6 BUS ARBITRATION

In order for the 8087 to read and write numeric data to and from memory, it must have a means of taking control of the local bus. With the 8086/88 this is accomplished through a request-grant exchange protocol. The 80186, however, makes use of HOLD/HOLD AC-

KNOWLEDGE protocol to exchange control of the bus with another processor. The 82188 supplies the necessary conversion to interface RQ/GT to HOLD/HLDA signals. The RQ/GT signal of the 8087 connects directly to the 82188's RQ/GT0 input while the 82188's HOLD and HLDA pins connect to the 80186's HOLD and HLDA pins.

When the 8087 requires control of the bus, the 8087 sends a request on the RQ/GT0 line to the 82188. The 82188 responds by sending a HOLD request to the 80186. When HLDA is received back from the 80186, the 82188 sends a grant back to the 8087 on the same RQ/GT0 line.

The 82188 also has provisions for adding a third bus-master to the system which uses HOLD/HLDA protocol. This is accomplished by using the 82188 SYSHOLD, SYSHLDA, and RQ/GT1 signals. The third processor requests the bus by pulling the SYSHOLD line high. The 82188 will route (and translate if necessary) the requests to the current bus master. If the 8087 has control, the 82188 will request control via the RQ/GT1 line which should be connected to the 8087's RQ/GT1 line.

The 8087 will relinquish control by getting off the bus and sending a grant pulse on the RQ/GT1 line. The 82188 responds by sending a SYSHLDA to the third processor. The third processor lowers SYSHOLD when it has finished on the bus. The 82188 routes this in the form of a release pulse on the RQ/GT1 line to the 8087. The 8087 then continues bus activity where it left off. The maximum latency from SYSHOLD to SYSHLDA is equal to the 80186 latency + 8087 latency + 82188 latency.

5.7 SPEED REQUIREMENTS

One of the most important timing specs associated with the 80186-8087 interface is the speed at which the system should run. The 8087 was designed to operate with a 33% duty cycle clock whereas the 80186 and 80188 were designed to operate with a 50% duty cycle clock. In order to run both parts off the same clock, the 8087 must run at a slower speed than is typically implied by its dash number in the 8086/88 family.

To determine the speed at which an 8087 may run (with a 50% duty cycle clock), the minimum low and high times of the 8087 must be examined. The maximum of these two minimum specs becomes the half-period of the 50% duty cycle system clock. For example, the 8087-1 provides worst case spec compatibility with the 80186 at system clock-speeds of up to 8 MHz. The clock waveforms are shown in Figure 12 using 10 MHz timings.

The minimum clock low time spec (T_{CLCH}) of the 10 MHz 8087 is 53 ns. The clock low time of an 8 MHz 80186 is specified to be:

$$\frac{1}{2}(T_{CLCL}) - 7.5$$

Solving for T_{CLCL} of the 80186 using T_{CLCH} of the 8087 yields the following:

$$\begin{aligned} \frac{1}{2}(T_{CLCL}) - 7.5 &= T_{CLCH} \\ (T_{CLCL}) &= 2(T_{CLCH} + 7.5) \\ T_{CLCL} &= 121 \text{ ns} \end{aligned}$$

The calculation shows minimum cycle time of the 80186 to be 121 ns. This time translates into a maximum frequency of 8.26 MHz.

6.0 BENCHMARKS

6.1 Introduction

The following benchmarks compare the overall system performance of an 8086, 80188, and an 80186 in numeric applications. Results are shown for all three processors in systems with the 8087 coprocessor and in systems using an 8087 software emulator. Three FORTRAN benchmark programs are used to dem-

onstrate the large increase in floating-point math performance provided by the 8087 and also the increase in performance due to the enhanced 80186 and 80188 host processors.

The 8086 results were measured on an Intel® Series III Microcomputer Development System with an iSBC® 86/12 board and an iSBC 337 multimodule. Typically, one wait state for memory read cycles and two wait states for memory write cycles are experienced in this environment.

The 80186 and 80188 results were measured on a prototype board which allowed zero wait state operation at 8 MHz. The benchmarks measured using the 8087 showed little sensitivity to wait states. Instructions executed on the 8087 tend to be long in comparison to the amount of bus activity required and, therefore, are not affected much by wait states.

The benchmarks measured using the software emulator are much more bus intensive and average from 10 to 15 percent performance degradation for one wait state.

All execution times shown here represent 8 MHz operation. The 8086 results were measured at 5 MHz and extrapolated to achieve 8 MHz execution times.

6.2 Interest Rate Calculations

Routines were written in FORTRAN-86 to calculate the final value of a fund given the annual interest and the present value. It is assumed that the interest will be compounded daily, which requires the calculation of the yearly effective rate. This value, which is the equivalent annual interest if the interest were compounded daily, is determined by the following formula:

$$\text{year} = (1 + (ir/np))^{np} - 1$$

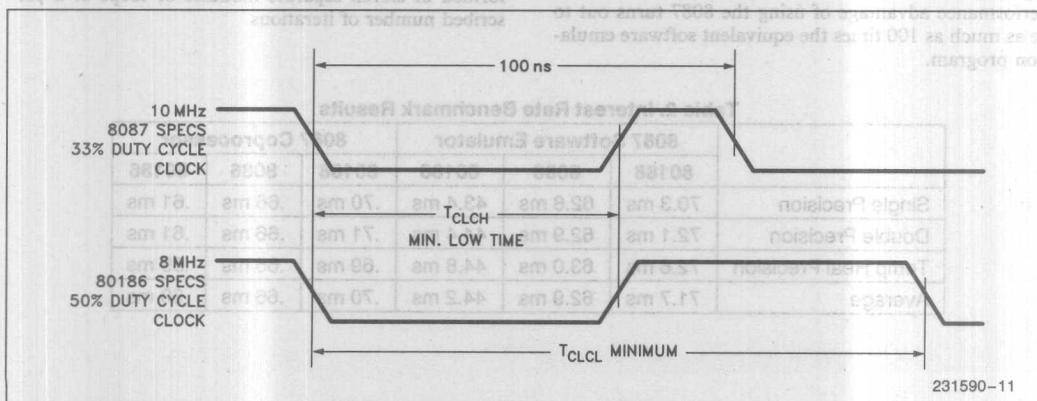


Figure 12. Clock Cycle Timing

where:
 yer is the yearly effective rate
 ir is the annual interest rate
 np is the number of compounding periods per annum

Once the yer is determined, the final value of the fund is determined by the formula:

$$fv = (1 + yer) * pv$$

where:
 pv is the present value
 fv is the future value

Results are obtained using single-precision, double-precision, and temporary real precision operands when:

ir is set to 10% (0.1)
 np is set to 365 (for daily compounding)
 pv is set to \$2,000,000

THE RESULTS:

	yer	Final Value
Single-Precision (32-bit)	10.514%	\$2,210,287.50
Double-Precision (64-bit)	10.516%	\$2,210,311.57
Temporary Real Precision	10.516%	\$2,210,311.57

The difference between the final single-precision and double-precision values is \$24.07; the difference in the final value between the double-precision and the temporary real precision is 0.000062 cents. Since the 8087 performs all internal calculations on 80-bit floating point numbers (temp real format), temporary real precision operations perform faster than single- or double-precision. No data conversions are required when loading or storing temporary real values in the 8087. Thus, for business applications, the double-precision computing of the 8087 is essential for accurate results, and the performance advantage of using the 8087 turns out to be as much as 100 times the equivalent software emulation program.

6.3 Matrix Multiply Benchmark Routine

A routine was written in FORTRAN-86 to compute the product of two matrices using a simple row/column inner-product method. Execution times were obtained for the multiplication of 32×32 matrices using double precision. The results of the benchmark are shown in Figure 14.

The results show the 8087 coprocessor systems performing from 23 to 31 times faster than the equivalent software emulation program. Both the 80188/87 and the 80186/87 systems outperform the 8086/87 system by 34 to 75 percent. This difference is mainly attributed to the fact that the matrix program largely consists of effective address calculations used in array accessing. The hardware effective address calculator of the 80186 and 80188 allow each array access to improve by as much as three times the 8086 effective address calculation.

6.4 Whetstone Benchmark Routine

The Whetstone benchmark program was developed by Harry Curnow for the Central Computer Agency of the British government. This benchmark has received high visibility in the scientific community as a measurement of main frame computer performance. It is a "synthetic" program. That is, it does not solve a real problem, but rather contains a mix of FORTRAN statements which reflect the frequency of such statements as measured in over 900 actual programs. The program computes a performance metric: "thousands of Whetstone instructions per second (KIPS)."

Simple variable and array addressing, fixed- and floating-point arithmetic, subroutine calls and parameter passing, and standard mathematical functions are performed in eleven separate modules or loops of a prescribed number of iterations.

Table 2. Interest Rate Benchmark Results

	8087 Software Emulator			8087 Coprocessor		
	80188	8086	80186	80188	8086	80186
Single Precision	70.3 ms	62.8 ms	43.4 ms	.70 ms	.66 ms	.61 ms
Double Precision	72.1 ms	62.9 ms	44.4 ms	.71 ms	.66 ms	.61 ms
Temp Real Precision	72.6 ms	63.0 ms	44.8 ms	.69 ms	.65 ms	.59 ms
Average	71.7 ms	62.9 ms	44.2 ms	.70 ms	.66 ms	.61 ms

The original coding of the Whetstone benchmark was written in Algol-60 and used single-precision values. It was rewritten in FORTRAN with single-precision values to exactly reflect the original intent. Another version was created using double-precision values. The results are shown in Table 3.

The results show the 8087 systems with the 80186 and 80188 outperforming the equivalent software emulation by 60 to 83 times. Additionally, the 80186 coupled with the 8087 outperformed the 8086/87 system by 22 percent.

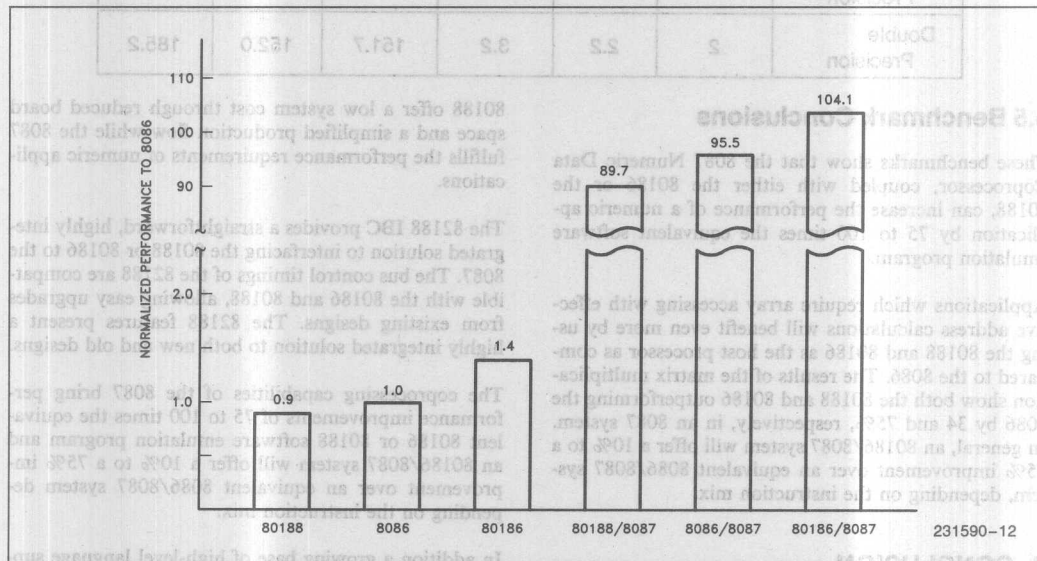


Figure 13. Interest Rate Benchmark Results

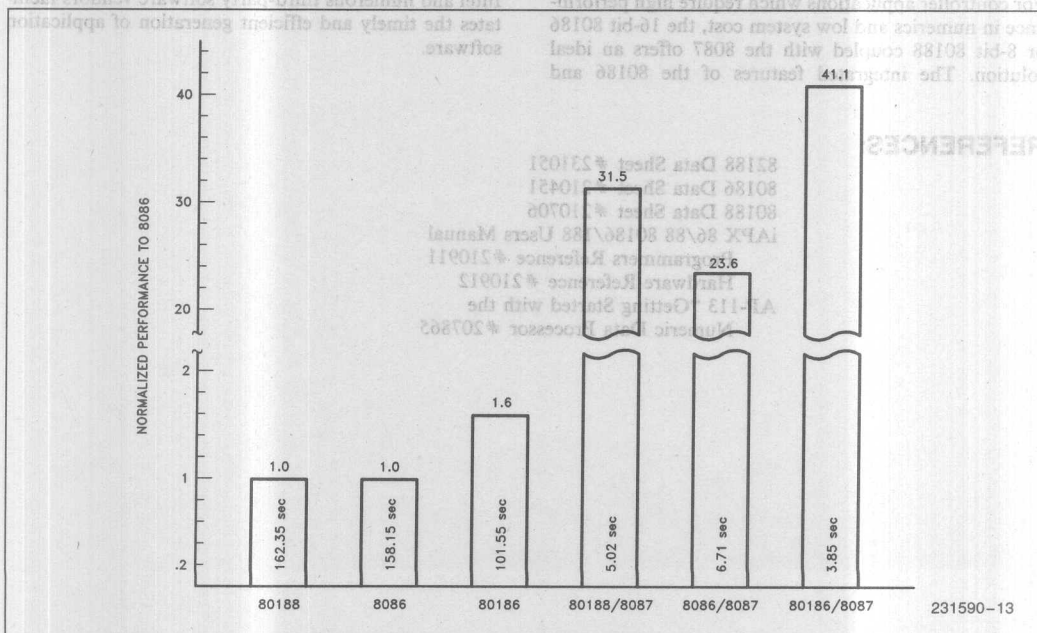


Figure 14. Double Precision Matrix Multiplication

Table 3. Whetstone Benchmark Results

Units = KIPS	8087 Software Emulator			8087 Coprocessor		
	80188	8086	80186	80188	8086	80186
Single Precision	2	2.3	3.3	165.8	178.0	197.6
Double Precision	2	2.2	3.2	151.7	152.0	185.2

6.5 Benchmark Conclusions

These benchmarks show that the 8087 Numeric Data Coprocessor, coupled with either the 80186 or the 80188, can increase the performance of a numeric application by 75 to 100 times the equivalent software emulation program.

Applications which require array accessing with effective address calculations will benefit even more by using the 80188 and 80186 as the host processor as compared to the 8086. The results of the matrix multiplication show both the 80188 and 80186 outperforming the 8086 by 34 and 75%, respectively, in an 8087 system. In general, an 80186/8087 system will offer a 10% to a 75% improvement over an equivalent 8086/8087 system, depending on the instruction mix.

80188 offer a low system cost through reduced board space and a simplified production flow while the 8087 fulfills the performance requirements of numeric applications.

The 82188 IBC provides a straightforward, highly integrated solution to interfacing the 80188 or 80186 to the 8087. The bus control timings of the 82188 are compatible with the 80186 and 80188, allowing easy upgrades from existing designs. The 82188 features present a highly integrated solution to both new and old designs.

The coprocessing capabilities of the 8087 bring performance improvements of 75 to 100 times the equivalent 80186 or 80188 software emulation program and an 80186/8087 system will offer a 10% to a 75% improvement over an equivalent 8086/8087 system depending on the instruction mix.

7. CONCLUSION

For controller applications which require high performance in numerics and low system cost, the 16-bit 80186 or 8-bit 80188 coupled with the 8087 offers an ideal solution. The integrated features of the 80186 and

In addition a growing base of high-level language support (FORTRAN, Pascal, C, Basic, PL/M, etc.) from Intel and numerous third-party software vendors facilitates the timely and efficient generation of application software.

REFERENCES:

- 82188 Data Sheet #231051
- 80186 Data Sheet #210451
- 80188 Data Sheet #210706
- iAPX 86/88 80186/188 Users Manual
- Programmers Reference #210911
- Hardware Reference #210912
- AP-113 "Getting Started with the Numeric Data Processor #207865

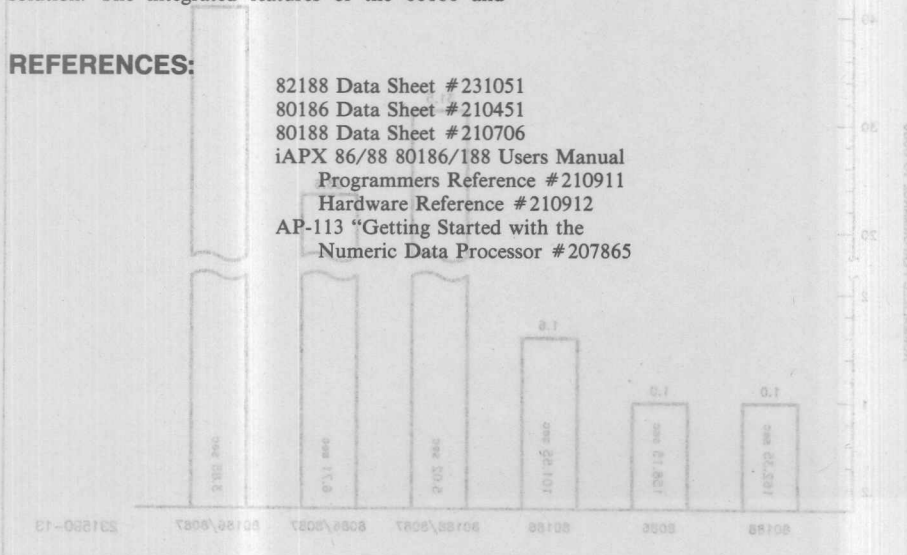


Figure 14. Double Precision Matrix Multiplication

This application note describes an efficient method of interfacing the 16-bit 80186 high integration microprocessor to the 80C21, 80C22, or the microcontroller-based 8044 serial communication controller. The interface hardware shown in Figure 1.1 is very simple and may be implemented with a programmable logic device or a gate-array. The 80186 and the microcontroller may run asynchronously and at different speeds. With this technique data transfers up to 200 Kbytes per second can be achieved between a 12 MHz microcontroller and an 8 MHz 80186.

The 8-bit 80188 high integration microprocessor can also be used with the same interface technique. The performance of the interface is the same since an 8-bit bus is used.

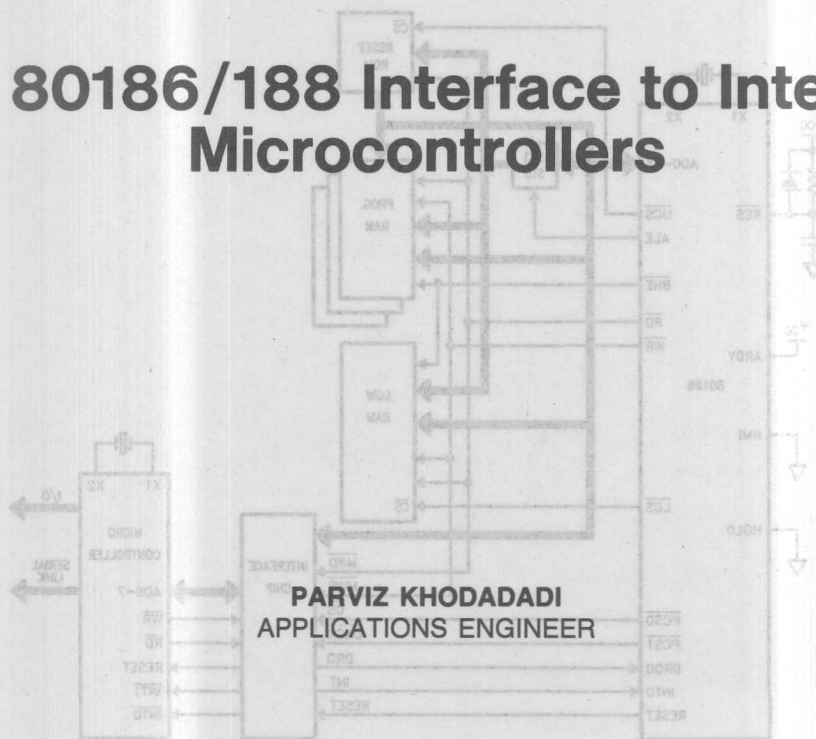
Interface to the 8044, 80C21, and the 80C22 is identical because they have identical pins (some pins have alternate functions). As an example, the software procedure for the 8044/80186 interface, which is the building block for the application driver, is supplied in this Application Note.

Systems which require I/O processing and serial data transmission are very software intensive. The communication task and I/O operations consume a lot of the system's intelligence and software. In many conventional systems the central processing unit carries the burden of all the communication and I/O operations in addition to its main routines, resulting in a slow and inefficient system.

In an ideal system, tasks are divided among processors to increase performance and achieve flexibility. One alternative solution is the combination of the Intel highly integrated 80186 microprocessor and the Intel 8-bit microcontroller, such as the 80C21, 80C22, or 8044. In this configuration, the 80186 provides the processing power

and the 8-bit microcontroller provides the intelligence for the I/O operations and data communication tasks. The 80186 runs application programs, performs the high level communication tasks, and provides the human interface. The microcontroller performs 8-bit math and single bit logic operations, the low level communication tasks, and I/O processing.

80186/188 Interface to Intel Microcontrollers



1.0 INTRODUCTION

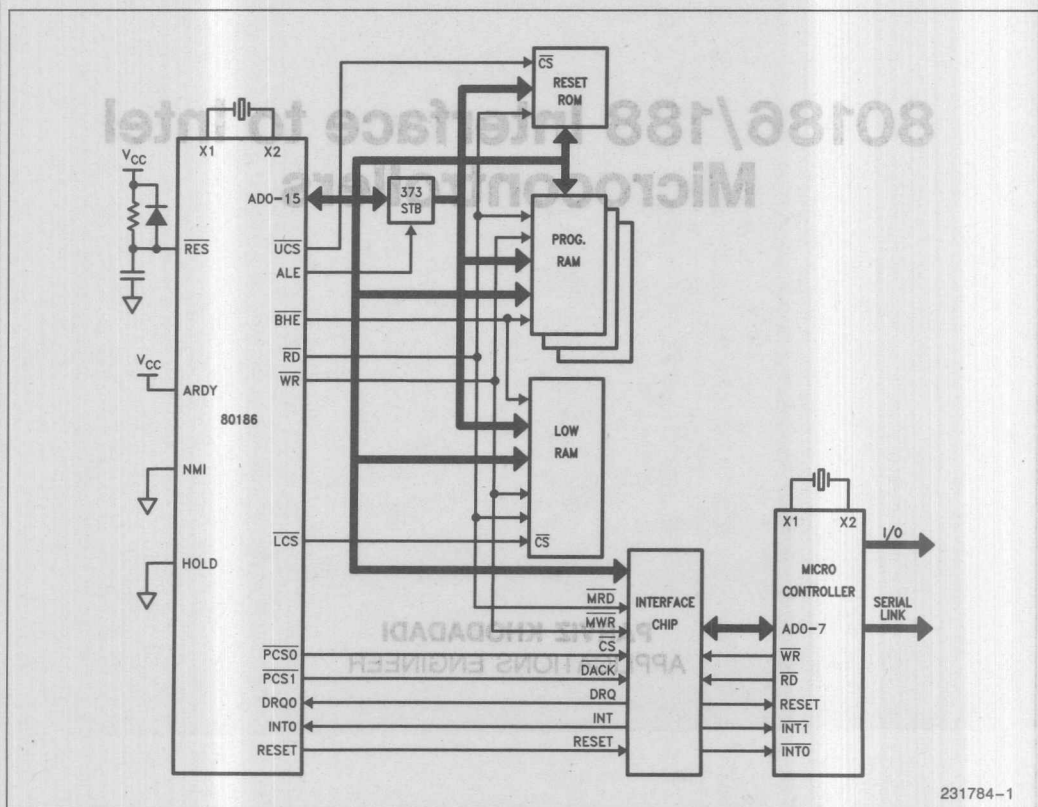
Systems which require I/O processing and serial data transmission are very software intensive. The communication task and I/O operations consume a lot of the system's intelligence and software. In many conventional systems the central processing unit carries the burden of all the communication and I/O operations in addition to its main routines, resulting in a slow and inefficient system.

In an ideal system, tasks are divided among processors to increase performance and achieve flexibility. One attractive solution is the combination of the Intel highly integrated 80186 microprocessor and the Intel 8-bit microcontrollers such as the 80C51, 8052, or 8044. In such a system, the 80186 provides the processing power and the 1 Mbyte memory addressability, while the controller provides the intelligence for the I/O operations and data communication tasks. The 80186 runs application programs, performs the high level communication tasks, and provides the human interface. The microcontroller performs 8-bit math and single bit boolean operations, the low level communication tasks, and I/O processing.

This application note describes an efficient method of interfacing the 16-bit 80186 high integration microprocessor to the 80C51, 8052, or the microcontroller-based 8044 serial communication controller. The interface hardware shown in Figure 1.1, is very simple and may be implemented with a programmable logic device or a gate-array. The 80186 and the microcontroller may run asynchronously and at different speeds. With this technique data transfers up to 200 Kbytes per second can be achieved between a 12 MHz microcontroller and an 8 MHz 80186.

The 8-bit 80188 high integration microprocessor can also be used with the same interface technique. The performance of the interface is the same since an 8-bit bus is used.

Interface to the 8044, 80C51, and the 8052 is identical because they have identical pinouts (some pins have alternate functions). As an example, the software procedures for the 8044/80186 interface, which is the building block for the application driver, is supplied in this Application Note.



231784-1

Figure 1.1. 80186/Microcontroller Based System

1.1 System Overview

The 80186 and the microcontrollers are processors. They each access memory and have address/data, read, and write signals. There are three common ways to interface multiple processors together:

- 1) First In First Out (FIFO)
- 2) Dual Port RAM (DPRAM)
- 3) Slave Port

The FIFO interface, compared to DPRAM, requires less TTL and is easier to interface; however, FIFOs are expensive. The DPRAM interface is also expensive and even more complex. When DPRAM is used, the address/data lines of each processor must be buffered, and hardware logic is needed to arbitrate access to DPRAM. The slave port interface given here is cheaper and easier than both FIFO and DPRAM alternatives.

The 80186 processor, when interfaced to this circuit, views the microcontroller as a peripheral chip with 8-bit data bus and no address lines (see Figure 1.1). It can read status and send commands to the microcontroller at any time. The microcontroller becomes a slave co-processor while keeping its processing power and serial communication capabilities.

The microcontrollers, with the interface hardware, have a high level command interface like many other data communication peripherals. For example, the 80186 can send the microcontroller commands such as Transmit or Configure. This means the designer does not have to write low level software to perform these tasks, and it offloads the 80186 to serve other functions in the application.

1.2 Application Examples:

The combination of the 80186 and a microcontroller basically provides all the functions that are needed in a system: a 16-bit CPU, 8-bit CPU, DMA controller, I/O ports, and a serial port. The 80C51 and the 8052 have an on-chip asynchronous channel, while the 8044 has an intelligent SDLC serial channel. In addition, many other functions such as timers, counters, and interrupt controllers are integrated in both the 80186 and the microcontrollers.

Applications of the system described above are in the area of robotics, data communication networks, or serial communication backplanes. A typical example is copiers. Different segments of the copy machine like the motor, paper feed, diagnostics, and error/warning displays are all controlled by microcontrollers. Each segment receives orders from and replies to the central processor which consists of the 80186 interfaced with a microcontroller.

Another common application is in the area of process controllers. An example is a central control unit for a multiple story building which controls the heating, cooling, and lighting of each room in each floor. In each room a microcontroller performs the above functions based on the orders received from the central processor. Depending on the throughput and type of the serial communication required, the 8044 or the 80C51 (8052) may be selected for the application.

2.0 OVERVIEW OF THE 80186, 80C51, 8052, AND 8044

This section briefly discusses the features of the microcontrollers and the 80186. For more information about these products please refer to the Intel Microcontroller and Microsystem components hand-books. Readers familiar with the above products may skip this section.

2.1 The 80186 Internal Architecture

The 80186 contains an enhanced version of Intel's popular 8086 CPU integrated with many other features common to most systems (Figure 2.1). The 16-bit CPU can access up to 1 Mbyte of memory and execute instructions faster than the 8086. With speed selection of 8, 10, and 12.5 MHz, this highly integrated product is the most popular 16-bit microprocessor for embedded control applications.

The on-chip DMA controller has two channels which can each be shared by multiple devices. Each channel is capable of transferring data up to 3.12 Mbytes per second (12.5 MHz speed). It offers the choice of byte or word transfer. It can be programmed to perform a burst transfer of a block of data, transfer data per specified time interval, or transfer data per external request.

The on-chip interrupt controller responds to both external interrupts and interrupts requested by the on-chip peripherals such as the timers and the DMA channels. It can be configured to generate interrupt vector addresses internally like the microcontrollers or externally like the popular 8259 interrupt controller. It can be configured to be a slave controller to an external interrupt controller (iRMX 86 mode) or be master for one or two 8259s which in turn may be masters for up to 8 more 8259s. When configured in master mode, each channel can support up to 64 external interrupts (128 total).

Three 16-bit timers are also integrated on the chip. Timer 0 and timer 1 can be configured to be 16-bit counters and count external events. If configured as timers, they can be started by software or by an external event. Timer 0 and 1 each contain a timer output pin. Transitions on these pins occur when the timers reach one of the two possible maximum counts. Timer

2 can be used as a prescaler for timer 0 and 1 or can be used to generate DMA requests to the on-chip DMA channel.

Finally, the integrated clock generator, the wait state generator, and the chip select logic reduce the external logic necessary to build a processing system.

2.2 The MCS-51 Internal Architecture

The 80C51BH, as shown in Figure 2.2, consists of an 8-bit CPU which can access up to 64 Kbytes of data memory (RAM) and 64 Kbytes of program memory (ROM). In addition, 4 Kbytes of ROM and 128 bytes of RAM are built onto the chip.

The on-chip interrupt controller supports five interrupts with two priority levels. There are two timers integrated in the 80C51. Timer 0 and 1 can be configured as 8-bit or 16-bit timers or event counters.

Finally the integrated full duplex asynchronous serial channel provides the human interface or communica-

tion capability with other microcontrollers. The UART supports data rates up to 500 kHz (with 15 MHz crystal) and can distinguish between address bytes and data bytes.

The 8052 has the same features as the 80C51 except it has 8 Kbytes of on-chip ROM and 256 bytes of on-chip RAM. In addition the 8052 has another timer which may be configured as the baud rate generator for the serial port.

2.3 The 8044 Internal Architecture

The 8044 has all the features of the 80C51. In addition the on-chip RAM size is increased to 192 bytes and an intelligent HDLC/SDLC serial channel (SIU) replaces the 80C51 serial port (see Figure 2.3). It supports data rates up to 2.4 Mbps when an external clock is used and 375 Kbps when the clock is extracted from the data line. The serial port can be used in half duplex point to point, multipoint, or one-way loop configurations.

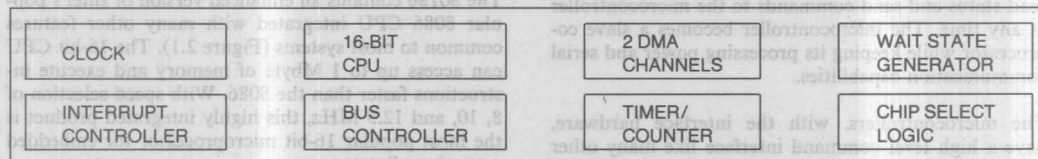


Figure 2.1. 80186 Block Diagram

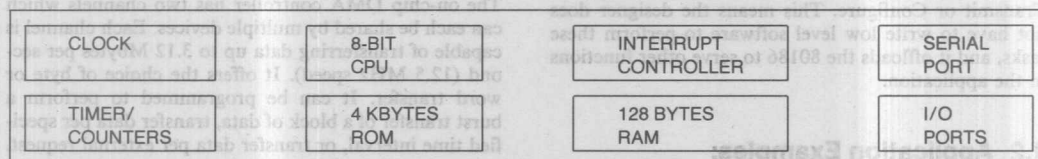


Figure 2.2. 80C51 Block Diagram

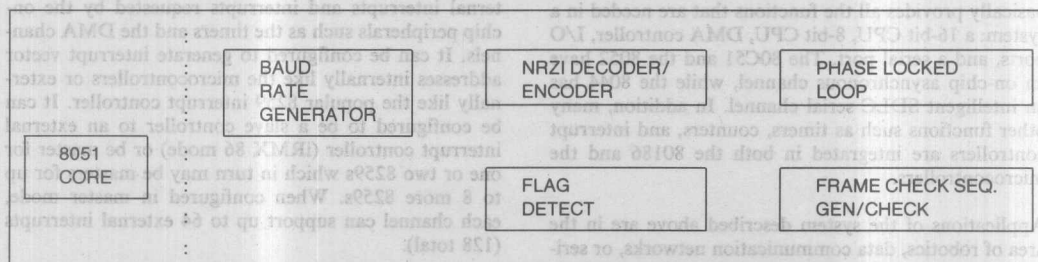


Figure 2.3. 8044 Block Diagram

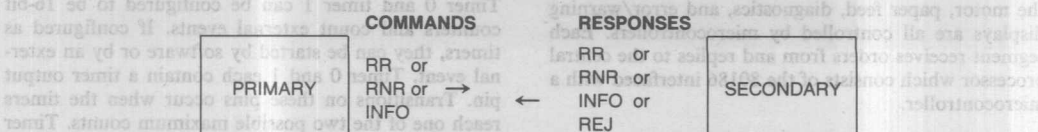


Figure 2.4. 8044 Automatic Response to SDLC Commands

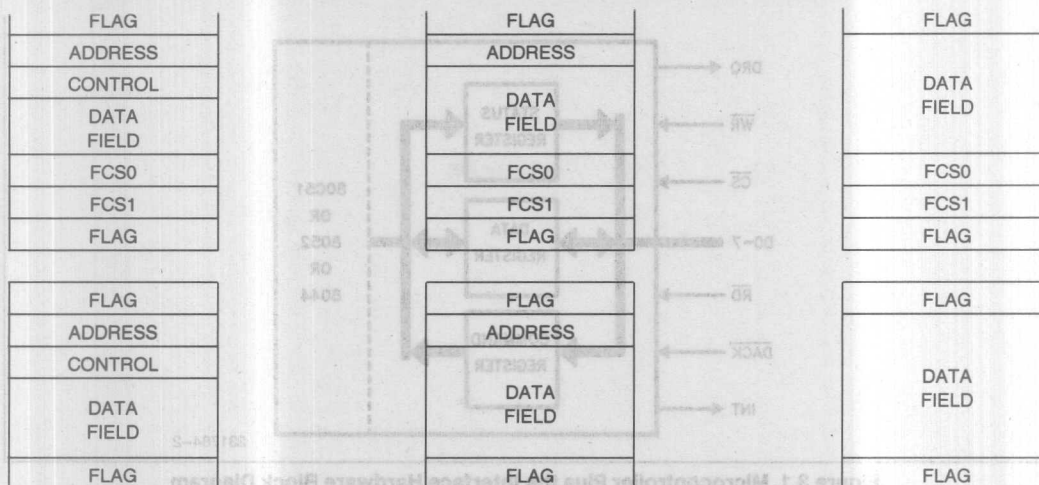


Figure 2.5. The 8044 Frame Formats

The SIU is called an intelligent channel because it responds to some SDLC commands automatically without the CPU intervention when it is set in auto mode. These automatic responses substantially reduce the communication software. Figure 2.4 gives the commands and the automatic responses.

The 8044 supports many types of frames including the standard SDLC format. Figure 2.5 shows the types of frames the 8044 can transmit and receive. If a format with an address byte is chosen, the 8044 performs address filtering during reception and transmits the contents of the station address register during transmission automatically. If a format with FCS bytes is chosen, the 8044 performs Cyclic Redundancy Check (CRC) during reception and calculates the FCS bytes during transmission of a frame in hardware. Two preamble bytes (PFS) may optionally be added to the frames. Formats that include the station address and the control byte are supported both in the auto and flexible modes.

3.0 80186/MICROCONTROLLER INTERACTION

The 80186 communicates with the microcontroller (8044, 80C51 or 8052) through the system's memory and the Command/Data and Status registers. The CPU creates a data structure in the memory, programs the DMA controller with the start address and byte count of the block, and issues a command to the microcontroller. A hypothetical block diagram of a microcontroller when used with the interface hardware is given in Figure 3.1.

Chip select and interrupt lines are used to communicate between the microcontroller and the host. The inter-

rupt is used by the microcontroller to draw the 80186's attention. The Chip Select is used by the 80186 to draw the microcontroller's attention to a new command.

There are two kinds of transfers over the bus: Command/Status and data transfers. Command/Status transfers are always performed by the CPU. Data transfers are requested by the microcontroller and are typically performed by the DMA controller.

The CPU writes commands using CS and WR signals and interrupts the microcontroller. The microcontroller reads the command, decodes it and performs the necessary actions. The CPU reads the status register using CS and RD signals (see Figure 4.1).

To initiate a command like TRANSMIT or CONFIGURE, a write operation to the microcontroller is issued by the CPU. A read operation from the CPU gives the status of the microcontroller. Section 5 discusses details on these commands and the status.

Any parameters or data associated with the command are transferred between the system memory and the microcontroller using DMA. The 80186 prepares a data block in memory. Its first byte specifies the length of the rest of the block. The rest of the block is the information field. The CPU programs the DMA controller with the start address of the block, length of the block and other control information and then issues the command to the microcontroller.

When the microcontroller requires access to the memory for parameter or data transfer, it activates the 80186 DMA request line and uses the DMA controller to achieve the data transfer. Upon completion of an operation, the microcontroller interrupts the 80186. The CPU then reads results of the operation and status of the microcontroller.

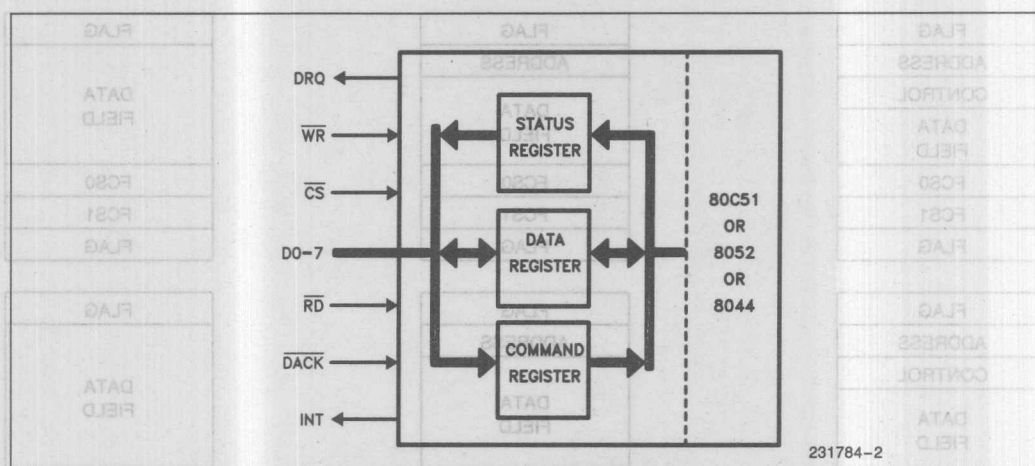


Figure 3.1. Microcontroller Plus the Interface Hardware Block Diagram

4.0 SYSTEM INTERFACE

There are two kinds of transfers over the bus: command/status and data transfers. The command/status transfers are always initiated and performed by the 80186. The data transfers are requested by the microcontroller using the DMA request (DRQ) line. In relatively slow systems the 80186 might also perform the data transfers. In that case, the request from the microcontroller will serve as an interrupt to the CPU. This mode of operation depends on the serial data rate.

The system interface performs command/status transfers, data/parameter transfers, and interrupts. This section describes the interface between the 80186 and a microcontroller shown in Figure 1.1. Section 6 describes the interface hardware.

4.1 Command/Status Transfers

The 80186 controls the microcontroller by writing into the command/data register and reading from the status register. The CPU writes a command by activating the chip select (PCSO), putting the command onto the data bus, and activating the WR signal. The command byte is latched into the command/data register, and the microcontroller is interrupted. In the interrupt service routine, the microcontroller reads the command byte from the command/data register, decodes the command byte, and activates the DRQ for data or parameter transfer if the decoded command requires such transfer.

At the end of parameter transfer the microcontroller updates the status register and interrupts the 80186.

The 80186 DMA controller performs a byte transfer from the memory to the Command/Data register on the rising edge of the 80186 WR signal (MWR), which is activated by the DMA controller. Figure 4.2 shows the write timing.

4.2 Data/Parameter Transfer

Data/parameter transfers are controlled by a pair of REQUEST/ACKNOWLEDGE lines: DMA Request line (DRQ) and DMA Acknowledge line (DACK). Data and parameters are transferred via the Command/Data register to or from memory.

In order to request a transfer from memory, the microcontroller activates the DRQ pin. The DRQ signal goes active after a read operation by the microcontroller. In response, the 80186 DMA controller performs a byte transfer from the memory to the Command/Data register. Data is transferred on the bus and written into the Command/Data register on the rising edge of the 80186 WR signal (MWR), which is activated by the DMA controller. Figure 4.2 shows the write timing.

In order to request a transfer to memory, the microcontroller activates the DRQ signal and outputs the data into the Command/Data latch. When the microcontroller WR signal goes active, DRQ is set. In response, the DMA performs the data transfer and resets the DRQ signal. Figure 4.3 shows the read timing.

4.3 Interrupt

The microcontroller reports on completion of an event by updating the status register and raising the interrupt signal assuming this signal is initially low. The interrupt is cleared by the command from the CPU where

the INTERRUPT ACKNOWLEDGE bit is set (MD7). The INTA bit is the most significant bit of the command byte. Figure 4.4 and 4.5 show the interrupt timing. Note that it is the responsibility of the CPU to clear the interrupt in order to prevent a deadlock.

80186 Pin Name			Function
$\overline{\text{CS}}$	$\overline{\text{RD}}$	$\overline{\text{WR}}$	
1	X	X	No Transfer to/from Command/Status
0	1	1	
0	0	0	Illegal
0	0	1	Read from Status Register
0	1	0	Write to Command/Data Register
$\overline{\text{DACK}}$	$\overline{\text{RD}}$	$\overline{\text{WR}}$	
1	X	X	No Transfer
0	1	1	
0	0	0	Illegal
0	0	1	Data Read from DMA Channel
0	1	0	Data Write to DMA Channel

NOTE:

Only one of CS, DACK may be active at any time.

Figure 4.1. Data Bus Control Signals and Their Functions

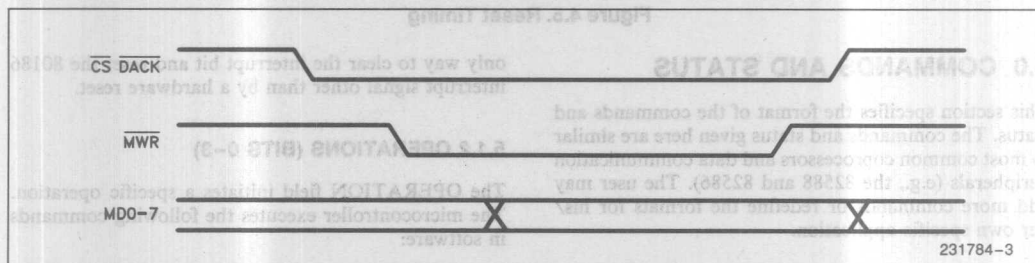


Figure 4.2. Write Timing

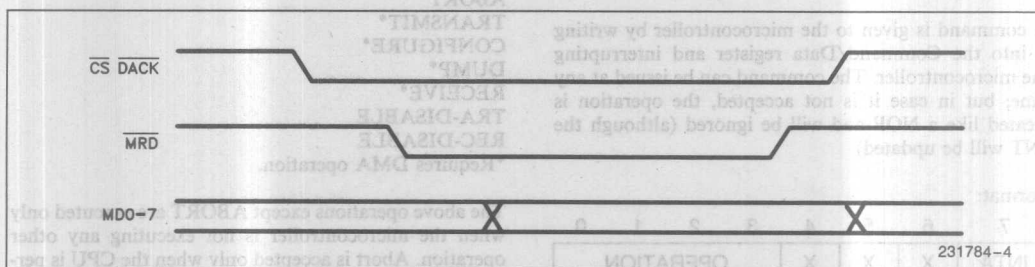


Figure 4.3. Read Timing

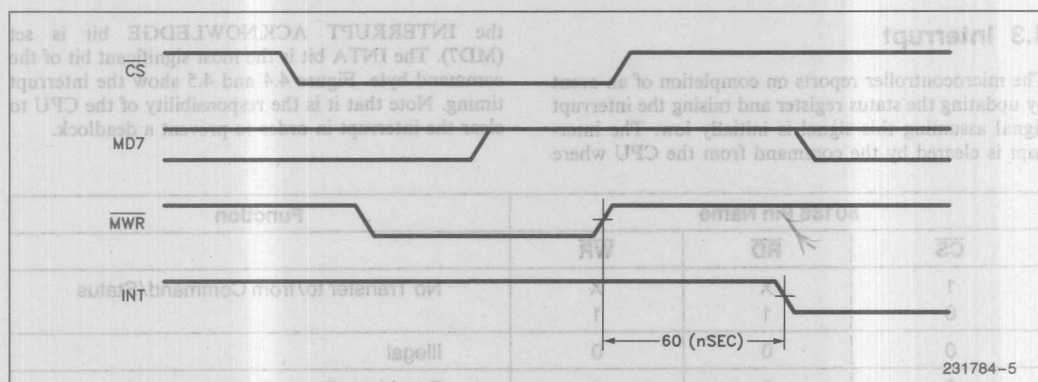


Figure 4.4. Interrupt Timing (Going Inactive)



Figure 4.5. Reset Timing

5.0 COMMANDS AND STATUS

This section specifies the format of the commands and status. The commands and status given here are similar to most common coprocessors and data communication peripherals (e.g., the 82588 and 82586). The user may add more commands or redefine the formats for his/her own specific application.

5.1 Commands

A command is given to the microcontroller by writing it into the Command/Data register and interrupting the microcontroller. The command can be issued at any time; but in case it is not accepted, the operation is treated like a NOP and will be ignored (although the INT will be updated).

Format:

7	6	5	4	3	2	1	0
INTA	X	X	X				OPERATION

5.1.1 ACKNOWLEDGING INTERRUPT (BIT 7)

The INTA bit, if set, causes the interrupt hardware signal and the interrupt bit to be cleared. This is the

only way to clear the interrupt bit and reset the 80186 interrupt signal other than by a hardware reset.

5.1.2 OPERATIONS (BITS 0-3)

The OPERATION field initiates a specific operation. The microcontroller executes the following commands in software:

NOP
ABORT
TRANSMIT*
CONFIGURE*
DUMP*
RECEIVE*
TRA-DISABLE
REC-DISABLE
*Requires DMA operation.

The above operations except ABORT are executed only when the microcontroller is not executing any other operation. Abort is accepted only when the CPU is performing a DMA operation.

Operations that require parameter transfer (e.g., CONFIGURE and DUMP) or data transfer (e.g., TRANSMIT and RECEIVE) are called parametric operations. The remaining are called non-parametric operations.

An operation is initiated by writing into the command register. This causes the microcontroller to execute the command decode instructions. Some of the operations cause the microcontroller to read parameters from memory. The parameters are organized in a block that starts with an 8-bit byte count. The byte count specifies the length of the rest of the block. Before beginning the operation, the DMA pointer of the DMA channel must point to the byte count. There is no restriction on the memory structure of the parameter block as long as the microcontroller receives the next byte of the block for every DMA request it generates. Transferring the bytes is the job of the 80186 DMA controller.

The microcontroller requests the byte-count and determines the length of the parameter block. It then requests the parameters.

Upon completion of the operation, (when interrupt is low) the microcontroller updates the status, raises the interrupt signal, and goes idle.

NOP

This operation does not affect the microcontroller. It has no parameters and no results.

ABORT

This operation attempts to abort the completion of an operation under execution. It is valid for CONFIGURE, TRANSMIT, DUMP, and RECEIVE. It is ignored for any of the above if transfer of parameters has already been accomplished. The microcontroller, upon reception of the ABORT command, stops the DMA operation and issues an Execution-Aborted interrupt.

TRANSMIT

This operation transmits one message. A message may be transmitted as an SDLC frame by the 8044, or in ASYNC protocol by the 80C51 or the 8052 serial port.

Figure 5.1 shows the format of the Transmit block. A typical transmit operation parameter block includes the destination address and the control byte in the information field. As an example, see the 8044 transmit block in Figure 7.2.

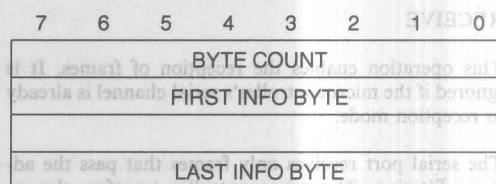


Figure 5.1. Format of Transmit Block

The transmit operation will either complete the execution or be aborted by a specific ABORT operation. A Transmit-Done or Execution-Aborted interrupt is issued upon completion of this operation.

CONFIGURE

This operation configures the microcontroller's internal registers. The length and the part of the configuration block that is modified are determined by the first two bytes of the command parameter (see Figure 5.2). The FIRST BYTE specifies the first register in the configure block that will be configured, and the BYTE COUNT specifies the number of registers that will be configured starting with the FIRST BYTE. For example, if the FIRST BYTE is 1 and the BYTE COUNT is the length of the configure block, then all of the registers are updated. If FIRST BYTE is 4 and BYTE COUNT is 2, then only the fourth register in the configure block is updated. Minimum byte count is 2.

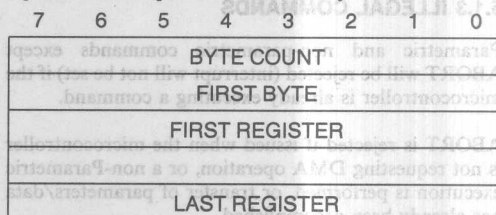


Figure 5.2. Format of Configure Block

A Configure-Done interrupt is issued when the operation is done unless ABORT was issued during the DMA operation.

DUMP

This operation causes dumping of a set of microcontroller internal registers to system memory. Figure 7.4 shows the format of the 8044 DUMP block.

The DUMP operation will either complete the execution or be aborted by a specific ABORT operation. A Dump-Done or Execution-Aborted interrupt is issued upon completion of this operation.

RECEIVE

This operation enables the reception of frames. It is ignored if the microcontroller's serial channel is already in reception mode.

The serial port receives only frames that pass the address filtering. The microcontroller transfers the received information and the byte count to the system memory using DMA. The completion of frame reception causes a Receive-Done event.

REC-DISABLE

This operation causes reception to be disabled. If transfer of data to the 80186 memory has already begun, then it is treated like the ABORT command. This operation has no parameters. REC-DISABLE is accepted only when the microcontroller's serial port is in receive mode.

TRA-DISABLE

This operation causes the transmission process to be aborted. If the microcontroller is fetching data from 80186 memory, then it is treated like the ABORT command. This operation has no parameters. It is accepted only when the serial port is in transmit mode.

5.1.3 ILLEGAL COMMANDS

Parametric and non-parametric commands except ABORT will be rejected (interrupt will not be set) if the microcontroller is already executing a command.

ABORT is rejected if issued when the microcontroller is not requesting DMA operation, or a non-Parametric execution is performed, or transfer of parameters/data has already been accomplished.

DMA operations shall not be aborted by any non-parametric or parametric command except by the ABORT command.

REC-DISABLE and TRA-DISABLE will not be accepted if the serial channel is idle.

5.2 Status

The microcontroller provides the information about the last operation that was executed, via the status register.

The microcontroller reports on these events by updating a status register and raising the INTERRUPT signal. Information from the status register is valid provided the interrupt signal is high or bit 0 of the status being read is set.

Format:

7	6	5	4	3	2	1	0
CTS*	RTS*	E	EVENT	DMA	INT		

*8044 only

5.2.1 INTERRUPT (BIT 0)

The interrupt bit is set together with the hardware interrupt signal. Setting the INT bit indicates the occurrence of an event. This bit is cleared by any command whose INTA bit is set. Status is valid only when this bit is set.

5.2.2 DMA OPERATION (BIT 1)

The DMA bit, when set, indicates that a DMA operation is in progress. This bit is set if the command received by the microcontroller requires data or parameter transfer. If this bit is clear, DRQ will be inactive. The DMA bit, when cleared, indicates the completion of a DMA operation.

5.2.3 ERROR (BIT 5)

The E bit, if set, indicates that the event generated for the operation that was completed contains a warning, or the operation was not accepted.

5.2.4 REQUEST TO SEND (BIT 6)

The RTS bit, if clear, indicates that the serial channel is requesting a transmission.

5.2.5 CLEAR TO SEND (BIT 7)

The CTS bit indicates that, if the RTS bit is clear, the serial port is active and transmitting a frame.

5.2.6 EVENT (BITS 2-4)

The event field specifies why the microcontroller needs the attention of the 80186.

The following events may occur:

CONFIGURE-DONE
TRANSMIT-DONE
DUMP-DONE
RECEIVE-DONE
RECEPTION-DISABLED
TRANSMISSION-DISABLED
EXECUTION-ABORTED

CONFIGURE-DONE
This event indicates the completion of a CONFIGURE operation.

TRANSMIT-DONE
This event indicates the completion of the TRANSMIT operation.

If the E bit is set, it indicates that the transmit buffer was already full.

DUMP-DONE
This event indicates that the DUMP operation is completed.

RECEIVE-DONE

This event indicates that a frame has been received and stored in memory. The format of the received message is indicated in Figure 5.3.

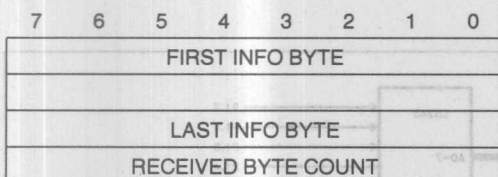


Figure 5.3. Format of Receive Block

Following the byte count, a few more bytes relating to the received frame such as the source address and the control byte may be transferred to the system memory using DMA. As an example, see the 8044 receive block in Figure 7.3.

Note that the format of a frame received by the microcontroller serial channel is configured by the CONFIGURE command.

If the E bit is set, buffer overrun has occurred.

RECEPTION-DISABLED

This event is issued as a result of a RCV-DISABLE operation that causes part of a frame to be disabled.

If the E bit is set, the serial port was already disabled, and the RCV-DISABLE is not accepted.

TRANSMISSION-DISABLED

This event is issued as a result of a TRA-DISABLE operation that causes transmission of a frame to be disabled.

The E bit, if set, indicates that the TRA-DISABLE operation was not accepted since the serial port was already idle, or transmission of a frame has already been accomplished.

EXECUTION-ABORTED

This event indicates that the execution of the last operation was aborted by the ABORT command.

If the E bit is set, ABORT was issued when the microcontroller was not executing any commands.

6.0 HARDWARE DESCRIPTION

The interface hardware shown in Figures 6.1 and 6.2 are identical. The difference is the status register. In Figure 6.2, an external latch is used to latch the status byte. This hardware is recommended if an extra I/O port on the microcontroller is required for some other applications, or external program and data memory is required for the microcontroller. The hardware shown in Figure 6.1 makes use of one of the microcontroller's I/O ports (Port 1) to latch the status to minimize hardware. The discussion of Sections 1 through 5 apply to both schematics.

6.1 Reset

After an 80186 hardware reset, the microcontroller is also reset. The on-chip registers are initialized as explained in the Intel Microcontroller Handbook. The reset signal also clears the 80186 interrupt and the microcontroller interrupt signals by resetting FF3 (Flip-Flop 3) and FF2 (Flip-Flop 2). Figure 4.5 shows the RESET timing.

6.2 Sending Commands

A bidirectional latched transceiver (74ALS646) is used for the Command/Data register. When the 80186 writes a command to the Command/Data register, it interrupts the microcontroller. The interrupt is generated only when bit 7 (INTA) of the command byte is set. When the 80186 PCS0 and WR signals go active to write the command, FF2 will be set and FF3 will be cleared. The output of FF3 is the interrupt to the 80186 and the INT status bit. The INT bit is cleared immediately to indicate that the status is no longer valid. The output of FF2 is the interrupt to the microcontroller. A high to low transition on this line will interrupt the microcontroller. The interrupt signal will be cleared as soon as the microcontroller reads the command from the Command/Data register.

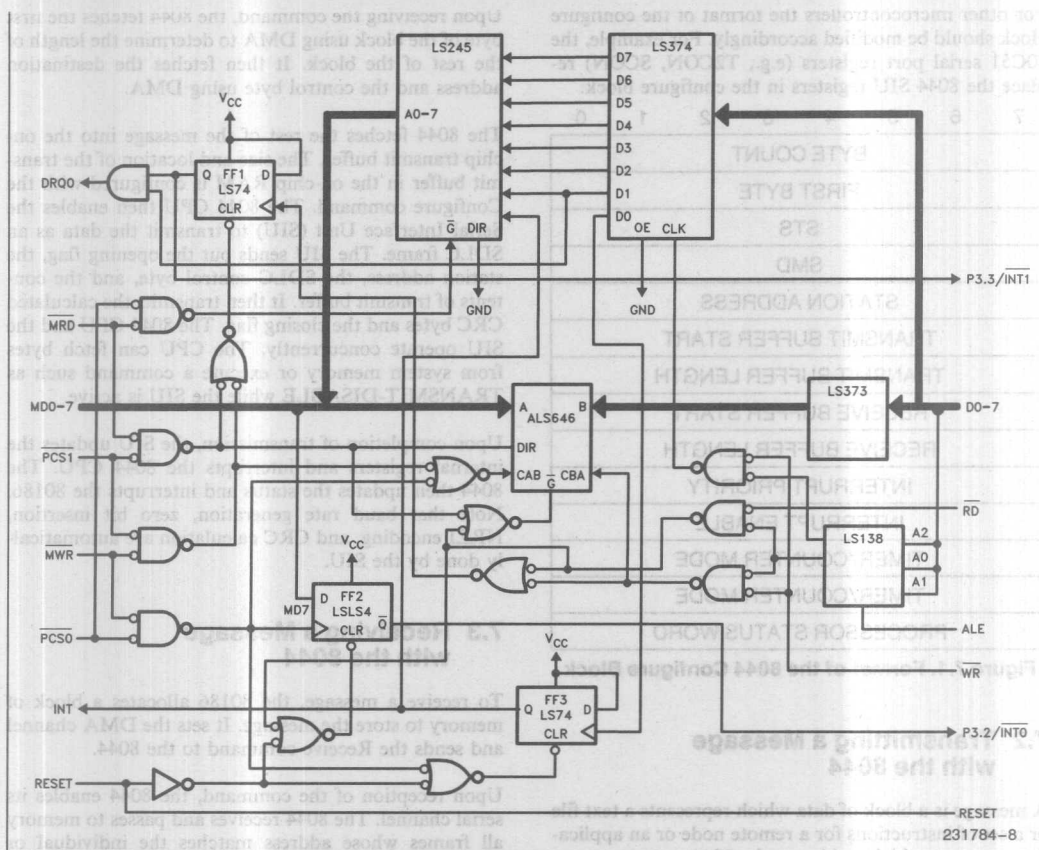


Figure 6.2. Hardware Interface

The microcontroller updates the status byte whenever a change occurs in the status and outputs the result to the status register. In order to read status, the 80186 activates the PCS0 line, and then activates the RD line. The contents of the status are put on the data bus, through the 74LS245 transceiver.

For systems that require two DMA channels, a second pair of DRQ1/DACK1 signals may easily be added to the hardware. In that case one of the status bits (DMA2) ANDed with the output of FF1 will serve as the second DMA request signal (DRQ1). DACK1 can be generated with the 80186 PCS2.

7.0 8044/80186 INTERFACE

This section shows how to make use of the status and commands described in section 5 and the hardware given in Figure 6.1 to interface the 80186 with the 8044. The 8044 code to implement these functions is shown in Appendix A.

7.1 Configuring the 8044

This operation configures the 8044 registers. The format of the configure block is shown in Figure 7.1. The part of the configuration block that is modified is determined by the first two bytes of the command parameter. The FIRST BYTE specifies the first register in the configure block that will be configured, and the BYTE COUNT specifies the number of registers that will be configured starting with the FIRST BYTE. For example, if the FIRST BYTE is 1 and the BYTE COUNT is 13, then all of the registers are updated. If FIRST BYTE is 4 and BYTE COUNT is 2, then transmit buffer start register is configured.

The configure command performs the following: 1) configures the interrupts and assigns their priorities; 2) assigns the start address and length of the transmit and receive buffers; 3) sets the station address; 4) sets the clock option and the frame format.

For other microcontrollers the format of the configure block should be modified accordingly. For example, the 80C51 serial port registers (e.g., T2CON, SCON) replace the 8044 SIU registers in the configure block.

7	6	5	4	3	2	1	0
BYTE COUNT							
FIRST BYTE							
STS							
SMD							
STATION ADDRESS							
TRANSMIT BUFFER START							
TRANSMIT BUFFER LENGTH							
RECEIVE BUFFER START							
RECEIVE BUFFER LENGTH							
INTERRUPT PRIORITY							
INTERRUPT ENABLE							
TIMER/COUNTER MODE							
TIMER/COUNTER MODE							
PROCESSOR STATUS WORD							

Figure 7.1. Format of the 8044 Configure Block

7.2 Transmitting a Message with the 8044

A message is a block of data which represents a text file or a set of instructions for a remote node or an application program which resides on the 8044 program memory. A message can be a frame (packet) by itself or can be comprised of multiple frames. An SDLC frame is the smallest block of data that the 8044 transmits. The 8044 can receive commands from the 80186 to transmit and receive messages. The 8044 on-chip CPU can be programmed to divide messages into frames if necessary. Maximum frame size is limited by the transmit or receive buffer.

To transmit a message, the 80186 prepares a transmit data block in memory as shown in Figure 7.2. Its first byte specifies the length of the rest of the block. The next two bytes specify the destination address of the node the message is being sent to and the control byte of the message. The 80186 programs the DMA controller with the start address of the block, length of the block and other control information and then issues the Transmit command to the 8044.

Upon receiving the command, the 8044 fetches the first byte of the block using DMA to determine the length of the rest of the block. It then fetches the destination address and the control byte using DMA.

The 8044 fetches the rest of the message into the on-chip transmit buffer. The size and location of the transmit buffer in the on-chip RAM is configured with the Configure command. The 8044 CPU then enables the Serial Interface Unit (SIU) to transmit the data as an SDLC frame. The SIU sends out the opening flag, the station address, the SDLC control byte, and the contents of transmit buffer. It then transmits the calculated CRC bytes and the closing flag. The 8044 CPU and the SIU operate concurrently. The CPU can fetch bytes from system memory or execute a command such as TRANSMIT-DISABLE while the SIU is active.

Upon completion of transmission, the SIU updates the internal registers and interrupts the 8044 CPU. The 8044 then updates the status and interrupts the 80186. Note that baud rate generation, zero bit insertion, NRZI encoding, and CRC calculation are automatically done by the SIU.

7.3 Receiving a Message with the 8044

To receive a message, the 80186 allocates a block of memory to store the message. It sets the DMA channel and sends the Receive command to the 8044.

Upon reception of the command, the 8044 enables its serial channel. The 8044 receives and passes to memory all frames whose address matches the individual or broadcast address and passes the CRC test.

The SIU performs NRZI decoding and zero bit deletion, then stores the information field of the received frame in the on-chip receive buffer. At the end of reception, the CPU requests the transfer of data bytes to 80186 memory using DMA. After transferring all the bytes, the 8044 transfers the data length, source address, and control byte of the received frame to the memory (see Figure 7.3). Upon completion of the transfers, the 8044 updates the status register and raises the interrupt signal to inform the 80186.

If the SIU is not ready when the first byte of the frame arrives, then the whole frame is ignored. Disabling reception after the first byte was passed to memory causes the rest of the frame to be ignored and an interrupt with Receive-Aborted event to be issued.

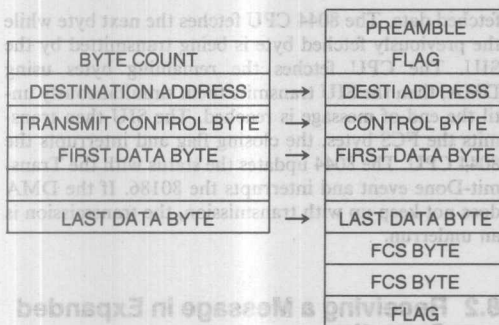


Figure 7.2. The 8044 Transmit Frame Structure and Location of Data Element in System Memory

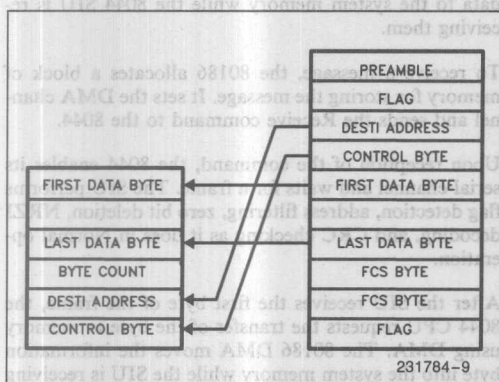


Figure 7.3. The 8044 Receive Frame Structure and Location of Received Data Element in System Memory

7.4 Dumping the 8044 Registers

Upon reception of the Dump command, the 8044 transfers the contents of its internal registers to the system memory (See Figure 7.4).

7	6	5	4	3	2	1	0
STS REG.							
SMD REG.							
STAD REG.							
TBS REG.							
TBL REG.							
TCB REG.							
RBS REG.							
RBL REG.							
RCB REG.							
RFL REG.							
PSW REG.							
IP REG.							
IE REG.							
TMOD REG.							
TCON REG.							

Figure 7.4. Format of the 8044 Dumped Registers

7.5 Aborting an Operation

To abort a DMA operation, the 80186 sends an Abort command to the Command/Data latch and interrupts the 8044. During a DMA operation, the 8044 puts the external interrupt to high priority; therefore, the Abort interrupt will suspend the execution of the operation in progress and update the status register with the Execution-Aborted event. It then returns the 8044 program counter to a location before the aborted operation started. The Abort software procedure given in Appendix A gives the details of the execution of the ABORT command.

7.6 Disabling the Transmission or Reception

Transmission of a frame is aborted if the 80186 sends a TRANSMIT-DISABLE command to the 8044. The command causes the 8044 to clear the Transmit Buffer

Full (TBF) bit. During transmission, if the TBF bit is cleared, the SIU will discontinue the transmission and interrupt the 8044 CPU.

The RECEIVE-DISABLE command causes the 8044 to clear the Receive Buffer Empty (RBE) bit. The SIU aborts the reception, if the RBE bit is cleared by the CPU.

When transmission or reception of a frame is discontinued, the SIU interrupts the 8044 CPU. The CPU then updates the status and interrupts the 80186.

7.7 Handling Interrupts

When the 80186 sends a command, it sets the 8044 external interrupt flag. The 8044 services the interrupt at its own convenience. In the interrupt service routine the 8044 executes the appropriate instructions for a given command. During execution of a command the 8044 ignores any command, except ABORT, sent by the 80186 (see section 5.1.2). This is accomplished by clearing the interrupt flag before the 8044 returns from the interrupt service routine. During DMA operations the 8044 sets the external interrupt to high priority. An interrupt with high priority can suspend execution of an interrupt service routine with low priority. The ABORT command given by the 80186 will interrupt the execution of the DMA transfer in progress. Upon completion of ABORT, execution of the last operation will not be resumed (see Appendix A). Note that any other command given during the DMA operation will also abort the operation in progress and should be avoided.

8.0 8044 IN EXPANDED OPERATION

To increase the number of information bytes in a frame, the 8044 can be operated in Expanded mode. In Expanded operation the system memory can be used as the transmit and receive buffer instead of the 8044 internal RAM. AP-283, "Flexibility in Frame Size Operation with the 8044", describes Expanded operation in detail.

8.1 Transmitting a Message in Expanded Operation

In Expanded operation the 8044 transmits the frame while it is fetching the data from the system memory using DMA. An internal transmit buffer is not necessary. The system memory can be used as the transmit buffer by the 8044.

Upon receiving the Transmit command, the 8044 enables the SIU and fetches the first data byte from the Command/Data register. The SIU transmits the opening flag, station address, and the control byte if the frame format includes these fields. It then transmits the

fetches data. The 8044 CPU fetches the next byte while the previously fetched byte is being transmitted by the SIU. The CPU fetches the remaining bytes using DMA, then the SIU transmits them simultaneously until the end of message is reached. The SIU then transmits the FCS bytes, the closing flag and interrupts the 8044 CPU. The 8044 updates the status with the Transmit-Done event and interrupts the 80186. If the DMA does not keep up with transmission, the transmission is an underrun.

8.2 Receiving a Message in Expanded Operation

In Expanded operation the DMA controller transfers data to the system memory while the 8044 SIU is receiving them.

To receive a message, the 80186 allocates a block of memory for storing the message. It sets the DMA channel and sends the Receive command to the 8044.

Upon reception of the command, the 8044 enables its serial channel and waits for a frame. The SIU performs flag detection, address filtering, zero bit deletion, NRZI decoding, and CRC checking as it does in Normal operation.

After the SIU receives the first byte of the frame, the 8044 CPU requests the transfer of the byte to memory using DMA. The 80186 DMA moves the information byte into the system memory while the SIU is receiving the next byte. The next byte is transferred to the memory after the SIU receives it. When the entire frame is received, the SIU checks the received Frame Check Sequence bytes. If there is no CRC error, the SIU updates the 8044 registers and interrupts the 8044 CPU. The CPU updates the status and interrupts the 80186.

9.0 CONCLUSION

This application note describes an efficient way to interface the 80186 and the 80188 microprocessors to the Intel 8-bit microcontrollers like the 80C51, 8052, and 8044. To illustrate this point the 80186 microprocessor interface to the 8044 microcontroller based serial communication chip was described. The hardware interface given here is very general and can interface the 8-bit microcontrollers to a variety of Intel microprocessors and DMA controllers. The microcontrollers with this interface hardware have the same benefits as both the Intel UPI-41/42 family and data communication peripheral chips such as the 82588 and the 82568 LAN controllers. Like the Intel UPI chips, they can be easily interfaced to microprocessors, and like the data communication peripherals, they execute high level commands. A similar approach can be used to interface Intel microprocessors to the 16-bit 8096 microcontroller.

APPENDIX A SOFTWARE

The software modules shown here implement the execution of commands and status explained in sections 5 and 7. The 80186 software provides procedures to send commands and read status. The 8044 software decodes and executes the commands, updates the status, and interrupts the 80186. The procedures given here are called by higher level software drivers. For example, an 80186 application program may use the Transmit command to send a block of data to an application program that resides in the 8044 ROM or in another remote node. The application programs and the drivers that perform the communication tasks run asynchronously since all communication tasks are interrupt-driven.

Figure A-1 shows how to assign the ports and control registers for an 80186-based system. The software is written for an Intel iSBC® 186/51 computer board. The 8044 hardware is connected to the computer board iSBX™ connector.

Figure A-2 shows the 80186 command procedures. These procedures are used by the data link driver.

Figure A-3 shows how the DMA controller is loaded and initialized for data and parameter transfer from the 80186 memory to the 8044. This procedure is used by the TRANSMIT and CONFIGURE commands.

Figure A-4 shows how the DMA controller is loaded and initialized for data and parameter transfer from the 8044 to the 80186 memory. This procedure is used by the RECEIVE and DUMP commands.

Figure A-5 shows an interrupt service routine which handles interrupts resulting from various events. Note that this routine is not complete. The user should write the software to respond to events.

Figure A-6 shows an example of the 80186 software. It shows how to start various operations. This is not a data link driver, but it gives the procedures needed to write a complete driver.

Figure A-7 shows how to initialize the 8044. The user application program should be inserted here.

Figures A-8 through A-13 show the 8044 external interrupt service routine. In this routine a command received from the 80186 is decoded, and one of the command procedures shown in Figures A-9 through A-13 is executed.

Figure A-14 shows the serial channel (SIU) interrupt service routine. Note that execution of TRANSMIT, RECEIVE, and TRANSMIT-DISABLE commands are completed in this routine.

Figure A-1 Port and Register Definitions for 80186 System

NAME COM_DRIVER

;** 80186 SOFTWARE FOR THE 80186/MICROCONTROLLER INTERFACE

;* 8044 BOARD CONNECTED TO THE SBX1 OF THE SBC 186/51 BOARD.
;* SBX1 INTO TIED TO 80130 IR[0-7]. CONNECT JUMPER 30 TO 46.
;* 80186 DMA CHANNEL 1 USED. CONNECT JUMPER 202 TO 203.

TRUE EQU 0FFFFH
FALSE EQU 0H

; 8044 REGISTERS

CMD_44 EQU 080H ; ADDRESS OF THE COMMAND REGISTER
ST_44 EQU 080H ; ADDRESS OF THE STATUS REGISTER
DATA_44 EQU 0D4H ; ADDRESS OF THE DATA REGISTER

; EVENTS

CON_DONE EQU 01H ; CONFIGURE DONE
TRA_DONE EQU 02H ; TRANSMIT DONE
DUM_DONE EQU 03H ; DUMP DONE
REC_DONE EQU 04H ; RECEIVE DONE
REC_DISA EQU 05H ; RECEPTION DISABLE
TRA_DISA EQU 06H ; TRANSMISSION DISABLE
ABO_DONE EQU 07H ; EXECUTION_ABORTED 231784-10

; COMMANDS (INTA=1)

ABO_CMD EQU 080H ; ABORT
REC_DIS_CMD EQU 081H ; RECEIVE DISABLE
XMIT_DIS_CMD EQU 082H ; TRANSMIT DISABLE
REC_CMD EQU 083H ; RECEIVE
TRA_CMD EQU 084H ; TRANSMIT
DUM_CMD EQU 085H ; DUMP
CON_CMD EQU 086H ; CONFIGURE
NOP_CMD EQU 087H ; NOP

; 80186 DMA CHANNEL 1 REGISTERS

SL_DMA1 EQU 0FFD0H ; SOURCE ADDRESS (LO WORD)
SH_DMA1 EQU 0FFD2H ; SOURCE ADDRESS (HI WORD)
DL_DMA1 EQU 0FFD4H ; DESTINATION ADDRESS (LO WORD)
DH_DMA1 EQU 0FFD6H ; DESTINATION ADDRESS (HI WORD)
CNT_DMA1 EQU 0FFD8H ; TRANSFER COUNT ADDRESS
CTL_DMA1 EQU 0FFDAH ; CONTROL ADDRESS

; 80186 INTERRUPT CONTROLLER REGISTERS

CTL0_INTR EQU 0FF38H ; INT 0 CONTROL ADDRESS
CTL1_INTR EQU 0FF3AH ; INT 1 CONTROL REGISTER
MASK_INTR EQU 0FF28H ; INT MASK REGISTER
EOI_INTR EQU 0FF22H ; INT EOI REGISTER
NSPEC_BIT EQU 08000H ; NON-SPECIFIC EOI

; 80130 INTERRUPT CONTROLLER REGISTERS

EOI_SINTR EQU 0E0H ; INT EOI REGISTER
MASK_SINTR EQU 0E2H ; MASK REGISTER
RD_IRR EQU 010H ; COMMAND TO 80130 TO READ IRR REG
RD_ISR EQU 011H ; COMMAND TO 80130 TO READ ISR REG
IV_BASE EQU 20H ; BASE OF 80130 INT CONTROLLER VECTOR

231784-11

Figure A-1. Port and Register Definitions for 80186 System


```

;*****
; INTERRUPT TABLE
;*****

INTERRUPTS    SEGMENT AT 0

                ORG (IV_BASE+1)*4H

IV_INTRO     LABEL  DWORD 0000H; IRI VECTOR

INTERRUPTS    ENDS

;*****

STACK         SEGMENT STACK 'STACK'

THE_STACK    DW 200H DUP(?)
TOS          LABEL WORD

STACK        ENDS

;*****

DATA         SEGMENT PUBLIC 'DATA'

REC_BUFFER   DB 1024 DUP(?)

CON_BUFFER   DB 08H,01H,00H,0D0H,55H,20H,05H,30H,05H

DUM_BUFFER   DB 0FH DUP(?)

TRA_BUFFER   DB 07H,55H,11H,01H,02H,03H,04H,05H

CMND_FLAG    DW FALSE

DATA        ENDS

```

231784-12

Figure A-1. Port and Register Definitions for 80186 System (Continued)

```

;*****
CODE         SEGMENT PUBLIC 'CODE'

ASSUME CS:CODE, DS:DATA, ES:NOTHING, SS:STACK

;*****

RCV_COMMAND  PROC FAR

    PUSH BP
    MOV BP,SP
    LES SI,DWORD PTR [BP+6] ; LOAD BUFFER POINTER
    MOV AX,WORD PTR [BP+10] ; LOAD BUFFER SIZE
    MOV AH,0H
    CALL REC_DMA ; CALL REC-DMA
    MOV AL,RCV_CMD ; LOAD RECEIVE COMMAND
    OUT CMD_44,AL ; SEND TO COMMAND/DATA REG
    POP BP
    RET

RCV_COMMAND  ENDP

;*****

XMIT_COMMAND PROC FAR

    PUSH BP
    MOV BP,SP
    LES SI,DWORD PTR [BP+6] ; LOAD BUFFER POINTER
    MOV AX,WORD PTR [BP+10] ; LOAD BUFFER SIZE
    MOV AH,0H
    CALL TRA_DMA ; CALL TRA-DMA
    MOV AL,XMIT_CMD ; LOAD TRANSMIT COMMAND
    OUT CMD_44,AL ; SEND TO COMMAND/DATA REG
    POP BP
    RET

XMIT_COMMAND ENDP

```

231784-13

Figure A-2. Setup and Execution of Commands

```

;*****
CONF_COMMAND PROC FAR
    PUSH BP
    MOV BP,SP
    LES SI,DWORD PTR[BP+6] ; LOAD BUFFER POINTER
    MOV AX,WORD PTR[BP+10] ; LOAD BUFFER SIZE
    MOV AH,0H
    CALL TRA_DMA ; CALL TRA-DMA
    MOV AL,CON_CMD ; LOAD CONFIGURE COMMAND
    OUT CMD_44,AL ; SEND TO COMMAND/DATA REG
    POP BP
    RET
CONF_COMMAND ENDP
;*****
DUMP_COMMAND PROC FAR
    PUSH BP
    MOV BP,SP
    LES SI,DWORD PTR[BP+6] ; LOAD BUFFER POINTER
    MOV AX,WORD PTR[BP+10] ; LOAD BUFFER SIZE
    MOV AH,0H
    CALL REC_DMA ; CALL REC-DMA
    MOV AL,DUM_CMD ; LOAD DUMP COMMAND
    OUT CMD_44,AL ; SEND TO COMMAND/DATA REG
    POP BP
    RET
DUMP_COMMAND ENDP
;*****
XMIT_DIS_COMMAND PROC FAR
    MOV AL,XMIT_DIS_CMD ; LOAD XMIT-DIS COMMAND
    OUT CMD_44,AL ; SEND TO COMMAND/DATA REG
    RET
XMIT_DIS_COMMAND ENDP
;*****
REC_DIS_COMMAND PROC FAR
    MOV AL,REC_DIS_CMD ; LOAD REC-DIS COMMAND
    OUT CMD_44,AL ; SEND TO COMMAND/DATA REG
    RET
REC_DIS_COMMAND ENDP
;*****
ABOR_COMMAND PROC FAR
    MOV AL,ABO_CMD ; LOAD ABORT COMMAND
    OUT CMD_44,AL ; SEND TO COMMAND/DATA REG
    RET
ABOR_COMMAND ENDP
;*****
NOP_COMMAND PROC FAR
    MOV AL,NOP_CMD ; LOAD NOP COMMAND
    OUT CMD_44,AL ; SEND TO COMMAND/DATA REG
    RET
NOP_COMMAND ENDP

```

231784-15

Figure A-2. Setup and Execution of Commands (Continued)

```

;*****
; ** RECEIVE DMA
; ARGES AX BUFFER SIZE
; ES:SI BUFFER POINTER
;

REC_DMA PROC NEAR
    MOV DX,CNT_DMA1 ; LOAD ADD OF TRANSFER COUNT REG
    OUT DX,AX ; PROGRAM TRANSFER COUNT REGISTER

    XOR BX,BX ; CLEAR BX
    MOV AX,ES ; LOAD SEG ADDRESS OF BUFFER
    SHL AX,1 ; CALCULATE LINEAR ADDRESS OF THE BUFFER
    RCL BX,1
    SHL AX,1
    RCL BX,1
    SHL AX,1
    RCL BX,1
    SHL AX,1
    RCL BX,1
    ADD AX,SI ; ADD THE OFFSET TO BASE
    ADC BX,0
    MOV DX,DL_DMA1 ; LOAD ADDRESS OF DEST POINTER (LO WORD)
    OUT DX,AX ; PROGRAM DEST POINTER REGISTER (LO WORD)
    MOV AX,BX
    MOV DX,DH_DMA1 ; LOAD ADDRESS OF DEST POINTER (HI WORD)
    OUT DX,AX ; PROGRAM DEST POINTER REGISTER (HI WORD)

    MOV AX,DATA_44 ; LOAD ADDRESS OF DATA REGISTER
    MOV DX,SL_DMA1 ; LOAD ADDRESS OF SOURCE POINTER
    OUT DX,AX ; PROGRAM SOURCE POINTER REGISTER (LO WORD)

    XOR AX,AX ; CLEAR AX
    MOV DX,SH_DMA1 ; LOAD ADDRESS OF SOURCE POINTER (HI WORD)
    OUT DX,AX ; PROGRAM SOURCE POINTER REGISTER (HI WORD)

    MOV DX,CTL_DMA1 ; LOAD ADDRESS OF CONTROL REGISTER
    MOV AX,1010001010100110B ; LOAD THE CONTROL WORD
    OUT DX,AX ; PROGRAM THE CONTRL REGISTER
    RET

REC_DMA ENDP

```

231784-16

Figure A-3. Loading and Starting the 80186 DMA Controller

```

;*****
; ** TRANSMIT DMA
; ARGES AX BUFFER SIZE
; ES:SI BUFFER POINTER
;

TRA_DMA PROC NEAR
    INC AX
    MOV DX,CNT_DMA1 ; LOAD ADD OF TRANSFER COUNT REG
    OUT DX,AX ; PROGRAM TRANSFER COUNT REGISTER

    XOR BX,BX ; CLEAR BX
    MOV AX,ES ; LOAD SEG ADDRESS OF BUFFER
    SHL AX,1 ; CALCULATE LINEAR ADDRESS OF THE BUFFER
    RCL BX,1
    SHL AX,1
    RCL BX,1
    SHL AX,1
    RCL BX,1
    SHL AX,1
    RCL BX,1
    ADD AX,SI ; ADD THE OFFSET TO BASE
    ADC BX,0
    MOV DX,SL_DMA1 ; LOAD ADDRESS OF SOURCE POINTER (LO WORD)
    OUT DX,AX ; PROGRAM SOURCE POINTER REGISTER (LO WORD)
    MOV AX,BX
    MOV DX,SH_DMA1 ; LOAD ADDRESS OF SOURCE POINTER (HI WORD)
    OUT DX,AX ; PROGRAM SOURCE POINTER REGISTER (HI WORD)

    MOV AX,DATA_44 ; LOAD ADDRESS OF DATA REGISTER
    MOV DX,DL_DMA1 ; LOAD ADDRESS OF DEST POINTER
    OUT DX,AX ; PROGRAM DEST POINTER REGISTER (LO WORD)

    XOR AX,AX ; CLEAR AX
    MOV DX,DH_DMA1 ; LOAD ADDRESS OF DEST POINTER (HI WORD)
    OUT DX,AX ; PROGRAM DEST POINTER REGISTER (HI WORD)

    MOV DX,CTL_DMA1 ; LOAD ADDRESS OF CONTROL REGISTER
    MOV AX,0001011010100110B ; LOAD THE CONTROL WORD
    OUT DX,AX ; PROGRAM THE CONTRL REGISTER
    RET

TRA_DMA ENDP

```

231784-17

Figure A-4. Loading and Starting the 80186 DMA Controller

```

;*****
; 80186 INTERRUPT ROUTINE
INT_186:
    PUSH AX
    PUSH DX
    MOV AX,NSPEC_BIT ; SEND NSPEC END OF INT
    MOV DX,EOI_INTR
    OUT DX,AX
    MOV AL,01100001B
    OUT EOI_INTR,AL
    IN AL,ST 44 ; READ THE STATUS
    AND AX,OFFH
; DECODE STATUS AND TAKE APPROPRIATE ACTION
    MOV DX,CTL_DMA1 ; DISABLE DMA
    IN AX,DX
    OR AX,0100B
    AND AX,NOT 010B
    OUT DX,AX
    MOV CHND_FLAG,TRUE
    POP DX
    POP AX
    IRET

```

231784-18

Figure A-5. Interrupt Service Routine

```

;*****
BEGIN:
    CLI
    CLD
; SET ALL REGISTERS SMALL MODEL
    MOV SP,DATA
    MOV DS,SP
    MOV ES,SP
    MOV SP,STACK
    MOV SS,SP
    MOV SP,OFFSET TOS
; SETUP INTERRUPT VECTORS
    PUSH ES
    XOR AX,AX
    MOV ES,AX
    MOV WORD PTR ES:IV_INTRO +0, OFFSET INT_186
    MOV WORD PTR ES:IV_INTRO +2, CS
    POP ES
SETUP 80130 INTERRUPT CONTROLLER
    MOV AL,00010011B ; ICW1
    OUT EOI_INTR,AL
    MUL AL
    MOV AL,IV_BASE ; ICW2
    OUT MASK_SINTR,AL
    MUL AL
    MOV AL,00000000B ; ICW4
    OUT MASK_SINTR,AL
    MUL AL
    MOV AL,OFCH ; MASK
    OUT MASK_SINTR,AL

```

231784-19

Figure A-6. Example of Executing Commands


```

MOV     AX,0000000000100000B
MOV     DX,CTLO_INTR
OUT     DX,AX

MOV     DX,CTLI_INTR
IN      AX,DX
OR      AX,0000000000101000B
OUT     DX,AX

MOV     AX,000EDH      ; MASK ALL BUT IO
MOV     DX,MASK_INTR
OUT     DX,AX
STI     ;ENABLE INTERRUPTS

;*** SEND CONFUIRE COMMAND

PUSH    WORD PTR CON_BUFFER      ; PUSH BUFFER SIZE
PUSH    DS                      ; PUSH BUFFER SEGMENT REGISTER
PUSH    OFFSET CON_BUFFER        ; PUSH OFFSET OF BUFFER
CALL    CONF_COMMAND            ; CALL CONFIGURE
ADD     SP,3*2

; WAIT FOR END OF COMMAND

WAIT1:  CMP     CMND_FLAG,TRUE
JNE     WAIT1
MOV     CMND_FLAG,FALSE

;*** SEND DUMP COMMAND

PUSH    WORD PTR DUM_BUFFER      ; PUSH BUFFER SIZE
PUSH    DS                      ; PUSH BUFFER SEGMENT REGISTER
PUSH    OFFSET DUM_BUFFER        ; PUSH OFFSET OF BUFFER
CALL    DUMP_COMMAND            ; CALL CONFIGURE
ADD     SP,3*2

WAIT2:  CMP     CMND_FLAG,TRUE
JNE     WAIT2
MOV     CMND_FLAG,FALSE

;*** SEND TRANSMIT COMMAND

PUSH    WORD PTR TRA_BUFFER      ; PUSH BUFFER SIZE
PUSH    DS                      ; PUSH BUFFER SEGMENT REGISTER
PUSH    OFFSET TRA_BUFFER        ; PUSH OFFSET OF BUFFER
CALL    XMIT_COMMAND            ; CALL COMMAND
ADD     SP,3*2

WAIT3:  CMP     CMND_FLAG,TRUE
JNE     WAIT3
MOV     CMND_FLAG,FALSE

;*** SEND RECEIVE COMMAND

PUSH    WORD PTR REC_BUFFER      ; PUSH BUFFER SIZE
PUSH    DS                      ; PUSH BUFFER SEGMENT REGISTER
PUSH    OFFSET REC_BUFFER        ; PUSH OFFSET OF BUFFER
CALL    REC_COMMAND            ; CALL COMMAND
ADD     SP,3*2

WAIT4:  CMP     CMND_FLAG,TRUE
JNE     WAIT4
MOV     CMND_FLAG,FALSE

CODE    ENDS
        BEGIN

```

231784-20

231784-21

Figure A-6. Example of Executing Commands (Continued)

```

$DEBUG NOMOD51
$INCLUDE (REG44.PDF)

; THE 8044 SOFTWARE DRIVER FOR THE 80186/8044 INTERFACE.

ORG 00H ; LOCATIONS 00 THRU 26H ARE USED
SJMP INIT ; BY INTERRUPT SERVICE ROUTINES.
ORG 03H ; VECTOR ADDRESS FOR EXT INTO.
JMP EINT0
ORG 23H ; VECTOR ADDRESS FOR SERIAL INT
JMP SIINT

***** INITIALIZATION *****
ORG 26H
INIT: MOV TCON,#00000001B ; EXT INTO: EDGE TRIGGER
MOV IE,#00010001B ; SI=EX0=1
CLR P1.1 ; CLEAR DRQ STATUS BIT
SETB EA ; ENABLE INTERRUPTS
DOT: SJMP DOT ; WAIT FOR AN INTERRUPT

```

231784-22

Figure A-7. Initialization Routine

```

*****EXTERNAL INTERRUPT 0 *****
EINT0: CLR P1.5 ; CLEAR THE E BIT
MOV DPTR,#100H ; LOAD DATA POINTER WITH A DUMMY NUMBER
MOVX A,@DPTR ; READ THE COMMAND BYTE.
ANL A,#00001111B ; KEEP THE OPERATION FIELD
MOV R2,A ; SAVE COMMAND

; DECODE COMMAND AND JUMP TO THE APPROPRIATE ROUTINE.
; COMMAND OPERATION (BITS0-3)
;
; ABORT 00H
; REC-DISABLE 01H
; TRA-DISABLE 02H
; RECEIVE 03H
; TRANSMIT 04H
; DUMP 05H
; CONFIGURE 06H
; NOP 07H

JNB PX0,J1
JMP CABO ; IF INTO IS SET TO PRIORITY 1,
; THEN DMA OPERATION WAS IN PROGRESS.
; EXECUTE ABORT REGARDLESS OF THE
; COMMAND ISSUED.
J1: CJNE A,#00H,J2
JMP CABO ; EXECUTE ABORT
; THIS LINE WILL BE EXECUTED IF ABORT WAS
; ISSUED WHEN THE 8044 IS NOT EXECUTING
; ANY COMMANDS.
J2: CJNE A,#01H,J3
JMP CRDIS ; EXECUTE RECEIVE-DISCONNECT
J3: CJNE A,#0B5H,J4
JMP CTDIS ; EXECUTE TRANSMIT-DISCONNECT
J4: CJNE A,#03H,J5
JMP CREC ; EXECUTE RECEIVE
J5: CJNE A,#04H,J6
JMP CTRA ; EXECUTE TRANSMIT
J6: CJNE A,#05H,J7
JMP CDUMP ; EXECUTE DUMP
J7: CJNE A,#06H,J8
JMP CCON ; EXECUTE CONFIGURE
J8: CJNE A,#07H,J9
JMP CNOP ; EXECUTE NOP
J9: RETI ; RETURN. OPERATION NOT RECOGNIZED.

```

231784-23

Figure A-8. External Interrupt Service Routine

; ** NOP COMMAND			
CNOP:	CLR	IE0	; IGNORE PENDING EXT INTO (IF ANY).
			; ANY INTERRUPT (COMMAD) DURING
			; EXECUTION OF AN OPERATION IS IGNORED
	RETI		; RETURN
; ** ABORT COMMAND			
CABO:	JNB	PX0,CABOJ1	; WAS DMA IN PROGRESS?
	CLR	PX0	; YES. EXT INTO: PRIORITY 0
	CLR	P1.1	; CLEAR DMA REQUEST
	SETB	P1.2	; UPDATE STATUS WITH
	SETB	P1.3	;ABORT-DONE EVENT
	SETB	P1.4	; (STATUS=DDH; E=0)
	CLR	IE0	; IGNORE PENDING EXT INTO (IF ANY).
	CLR	P1.0	; SET INT BIT AND INTERRUPT 80186
	SETB	P1.0	; WAIT TILL INTERRUPT IS ACKNOWLEDGED
	JB	P3.2,\$; EXECUTE THE NEXT "RETI" TWICE
	POP	ACC	; POP OUT THE OLD HI BYTE PC
	POP	ACC	; POP OUT THE OLD LOW BYTE PC
	MOV	B,#HIGH(\$+10)	; HI BYTE ADDRESS OF CABOJ2
	MOV	ACC,#LOW(\$+7)	; LOW BYTE ADDRESS OF CABOJ2
	PUSH	ACC	; PUSH THE ADDRESS OF THE NEXT
	PUSH	B	; "RETI" INSTRUCTION INTO STACK
CABOJ2:	RETI		; RETURN
CABOJ1:	NOP		; DMA WAS NOT IN PROGRESS
	SETB	P1.5	; SET THE E BIT
	SETB	P1.2	; UPDATE STATUS WITH
	SETB	P1.3	;ABORT-DONE EVENT
	SETB	P1.4	; (STATUS=FDH; E=1)
	CLR	IE0	; IGNORE PENDING EXT INTO (IF ANY).
	CLR	P1.0	; SET INT BIT AND INTERRUPT 80186
	SETB	P1.0	; WAIT TILL INTERRUPT IS ACKNOWLEDGED
	JB	P3.2,\$; RETURN
	RETI		

231784-24

Figure A-9. Execution of NOP and ABORT Commands

; ** CONFIGURE COMMAD			
CCON:	MOV	DPTR,#100H	
	CLR	IE0	; IGNORE PENDING EXT INTO (IF ANY)
	SETB	PX0	; EXT INTO: PRIORITY 1
			; PX0 IS SET TO ACCEPT ABORT
			; DURING DMA OPERATION.
	SETB	P1.1	; ENABLE DMA REQUEST
	JB	P3.3,\$; WAIT FOR DMA ACK.
	MOVX	A,@DPTR	; READ FROM COMMAD/DATA REGISTER
	MOV	RO,A	; LOAD BYTE COUNT
	DEC	RO	; DECREMENT BYTE COUNT
	JB	P3.3,\$; WAIT FOR DMA ACK.
	MOVX	A,@DPTR	; READ FROM COMMAND/DATA REGISTER
	MOV	R1,A	; LOAD FIRST-BYTE
	JB	P3.3,\$; WAIT FOR DMA ACK.
	MOVX	A,@DPTR	; READ FROM COMMAND/DATA REGISTER
	CJNE	R1,#01H,CCONJ1	; CHECK THE FIRST-BYTE
	MOV	STS,A	; UPDATE THE STS REGISTER
	INC	R1	; INC. POINTER TO THE CONF. BLOCK
	DJNZ	RO,CCONF4	; CHECK THE BYTE COUNT
	JMP	CCONT1	
CCONF4:	JB	P3.3,CCONF4	
	MOVX	A,@DPTR	
CCONJ1:	CJNE	R1,#02H,CCONJ2	
	MOV	SMD,A	
	INC	R1	
	DJNZ	RO,CCONF5	
	JMP	CCONT1	
CCONF5:	JB	P3.3,CCONF5	
	MOVX	A,@DPTR	
CCONJ2:	CJNE	R1,#03H,CCONJ3	
	MOV	STAD,A	
	INC	R1	
	DJNZ	RO,CCONF6	
	JMP	CCONT1	
CCONF6:	JB	P3.3,CCONF6	
	MOVX	A,@DPTR	
CCONJ3:	CJNE	R1,#04H,CCONJ4	

231784-25

Figure A-10. Execution of CONFIGURE Command

```

DJBZ R0, CCONF7
JMP CCONT1
CCONF7: JB P3.3, CCONF7
MOVX A, #DPTR
CCONJ4: CJNE R1, #05H, CCONJ5
MOV TBL, A
INC R1
DJBZ R0, CCONF8
JMP CCONT1
CCONF8: JB P3.3, CCONF8
MOVX A, #DPTR
CCONJ5: CJNE R1, #06H, CCONJ6
MOV RBS, A
INC R1
DJBZ R0, CCONF9
JMP CCONT1
CCONF9: JB P3.3, CCONF9
MOVX A, #DPTR
CCONJ6: CJNE R1, #07H, CCONJ7
MOV RBL, A
INC R1
DJBZ R0, CCONF8
JMP CCONT1
CCONF8: JB P3.3, CCONF8
MOVX A, #DPTR
CCONJ7: CJNE R1, #08H, CCONJ8
MOV IP, A
INC R1
DJBZ R0, CCONF8
JMP CCONT1
CCONF8: JB P3.3, CCONF8
MOVX A, #DPTR
CCONJ8: CJNE R1, #09H, CCONJ9
MOV IE, A
INC R1
DJBZ R0, CCONF8
JMP CCONT1
CCONF8: JB P3.3, CCONF8
MOVX A, #DPTR
CCONJ9: CJNE R1, #0AH, CCONJA
MOV TMD, A
INC R1
DJBZ R0, CCONF8
JMP CCONT1
CCONF8: JB P3.3, CCONF8
MOVX A, #DPTR
CCONJA: CJNE R1, #0BH, CCONJB
MOV TCON, A
INC R1
DJBZ R0, CCONF8
JMP CCONT1
CCONF8: JB P3.3, CCONF8
MOVX A, #DPTR
CCONJB: CJNE R1, #0CH, ERROR1
MOV PSW, A
INC R1
DJBZ R0, ERROR1
JMP CCONT1
ERROR1: NOP
SETB P1.5 ; ILLEGAL BYTE COUNT
; SET THE E STATUS BIT
CCONT1: NOP
CLR P1.1
CLR PX0 ; CLEAR DMA REQUEST
; EXT INTO: PRIORITY 0
SETB P1.2 ; UPDATE STATUS WITH
CLR P1.3 ; CONFIGURE-DONE EVENT
CLR P1.4 ; (STATUS=C5H IF E=0)
CLR IE0 ; IGNORE PENDING EXT INTO (IF ANY)
CLR P1.0
SETB P1.0 ; INTERRUPT THE 80186
JB P3.2, $ ; WAIT TILL INTERRUPT IS ACKNOWLEDGED
RETI ; RETURN

```

231784-27

Figure A-10. Execution of CONFIGURE Command (Continued)


```

; ** DUMP COMMAND
CDUMP:  MOV  A,STS      ; LOAD THE FIRST DUMP REG INTO ACC
        MOVX @DPTR,A   ; WRITE TO THE COMMAND/DATA REGISTER
        CLR  IE0       ; IGNORE PENDING EXT INTO (IF ANY)
        SETB PX0       ; INTERRUPT 0: PRIORITY 1
        SETB P1.1      ; ENABLE DMA REQUEST
        JB   P3.3,$     ; WAIT FOR DMA ACK
        MOV  A,SMD
        MOVX @DPTR,A
        JB   P3.3,$
        MOV  A,STAD
        MOVX @DPTR,A
        JB   P3.3,$
        MOV  A,TBS
        MOVX @DPTR,A
        JB   P3.3,$
        MOV  A,TBL
        MOVX @DPTR,A
        JB   P3.3,$
        MOV  A,TCB
        MOVX @DPTR,A
        JB   P3.3,$
        MOV  A,RBS
        MOVX @DPTR,A
        JB   P3.3,$
        MOV  A,RBL
        MOVX @DPTR,A
        JB   P3.3,$
        MOV  A,RCB
        MOVX @DPTR,A
        JB   P3.3,$
        MOV  A,RFL
        MOVX @DPTR,A
        JB   P3.3,$
        MOV  A,PSW
        MOVX @DPTR,A
        JB   P3.3,$
        MOV  A,IP
        MOVX @DPTR,A
        JB   P3.3,$
        MOV  A,IE
        MOVX @DPTR,A
        JB   P3.3,$
        MOV  A,TMOD
        MOVX @DPTR,A
        JB   P3.3,$
        MOV  A,TCON
        MOVX @DPTR,A
        JB   P3.3,$
        CLR  P1.1      ; DISABLE DRQ
        CLR  PX0       ; EXTERNAL INTO: PRIORITY 0
        SETB P1.2      ; UPDATE STATUS WITH
        SETB P1.3      ; DUMP-DONE EVENT
        CLR  P1.4      ; (STATUS=CDH)

        CLR  IE0       ; IGNORE PENDING EXT INTO
        CLR  P1.0      ; INTERRUPT THE 80186
        SETB P1.0
        JB   P3.2,$     ; WAIT TILL INTERRUPT IS ACKNOWLEDGED
        RETI          ; RETURN

```

Figure A-11. Execution of DUMP Command

```

; ** RECEIVE COMMAND.
CREC:  JNB  RBE,CRECJ1      ; IS SIU ALREADY IN RECEIVE MODE?
        SETB P1.5          ; YES. SET THE E BIT
CRECJ1: SETB  RBE          ; NO. ENABLE RECEPTION
        CLR  RBPTR         ; CLEAR RECEIVE BUFFER PROTECT BIT
        CLR  IEO           ; IGNORE PENDING EXT INTO (IF ANY)
        RETI              ; RETURN. UPDATE STATUS IN THE
                           ; SIU INTERRUPT ROUTINE.

; ** TRANSMIT COMMAND.
CTRA:  MOV  R1,TBS         ; LOAD TRANSMIT BUFFER START
        CLR  IEO           ; IGNORE PENDING EXT INTO (IF ANY)
        SETB PK0           ; EXT INTO: PRIORITY 1
        SETB P1.1         ; ENABLE DMA REQUEST
        JB   P3.3,$        ; WAIT FOR DMA ACK.
        MOVX A,@DPTR       ; READ FROM COMMAND/DATA REG.
        MOV  RO,A          ; LOAD THE BYTE COUNT
        DEC  A             ; SUBTRACT 2 FROM THE BYTE
        DEC  A             ; COUNT AND LOAD INTO XMIT
        MOV  TBL,A         ; LOAD BUFFER LENGTH
CTRAJ2: JB   P3.3,CTRAJ2    ; WAIT FOR DMA ACK.
        MOVX A,@DPTR       ; READ FROM COMMAND/DATA REG.
        MOV  STAD,A        ; LOAD DESTINATION ADDRESS
        DEC  RO            ; DECREMENT THE BYTE COUNT
CTRAJ3: JB   P3.3,CTRAJ3    ; WAIT FOR DMA ACK.
        MOVX A,@DPTR       ; READ FROM COMMAND/DATA REG.
        MOV  TCB,A         ; LOAD THE TRANSMIT CONTROL BYTE
        DJNZ RO,CTRAJ4     ; IS THERE ANY INFO. BYTE?
        SJMP CTRAJ5        ; NO.
CTRAJ4: JB   P3.3,CTRAJ4    ; YES. WAIT FOR DMA ACK.
        MOVX A,@DPTR       ; READ FROM COMMAND/DATA REG.
        MOV  @R1,A         ; MOVE DATA TO THE TRANSMIT BUFFER
        INC  R1            ; INC. POINTER TO BUFFER
        DJNZ RO,CTRAJ4     ; LAST BYTE FETCHED INTO THE BUFFER?
CTRAJ5: CLR  P1.1         ; NO. FETCH THE NEXT BYTE
        CLR  PK0          ; YES. DISABLE DMA REQUEST
        SETB TBF          ; EXT INTO: PRIORITY 0
        SETB RTS         ; SET TRANSMIT BUFFER FULL
        CLR  IEO         ; ENABLE TRANSMISSION
        RETI              ; IGNORE PENDING EXT INTO (IF ANY)
                           ; RETURN. UPDATE STATUS IN THE
                           ; SIU INTERRUPT ROUTINE

```

231784-30

Figure A-12. Execution of RECEIVE and TRANSMIT Commands

```

; ** TRANSMIT-DISCONNECT COMMAND
CTDIS:  JB   TBF,CTDIJ1     ; IS TRANSMIT BUFFER ALREADY EMPTY?
        SETB P1.5          ; YES. SET THE E BIT
CTDIJ1: CLR  TBF          ; NO. CLEAR TRANSMIT BUFFER
        CLR  IEO           ; IGNORE PENDING EXT INTO (IF ANY)
        RETI              ; RETURN. UPDATE STATUS IN THE
                           ; SIU INTERRUPT ROUTINE.

; ** RECEIVE-DISCONNECT COMMAND
CRDIS:  JB   RBE,CRDIJ1     ; IS RECEIVE BUFFER ALREADY EMPTY?
        SETB P1.5          ; YES. SET THE E BIT
CRDIJ1: CLR  RBE          ; NO. CLEAR RECEIVE BUFFER

        SETB P1.2         ; UPDATE STATUS WITH
        CLR  P1.3         ; RECEPTION-DISABLED EVENT
        SETB P1.4         ; (STATUS=D5 IF E=0)

        CLR  IEO
        CLR  P1.0
        SETB P1.0         ; INTERRUPT THE 80186
        JB   P3.2,$        ; WAIT TILL INTERRUPT IS ACKNOWLEDGED
        RETI              ; RETURN

```

231784-31

Figure A-13. Execution of RECEIVE-DISCONNECT and TRANSMIT-DISCONNECT Commands

```

;***** SERIAL CHANNEL (SIU) INTERRUPT *****
SIINT: CLR SI
      MOV A,R2 ; LOAD THE OPERATION FIELD
      CJNE A,#03H,SINTJ1 ; RECEIVE COMMAND PENDING?
      JMP SIREC ; YES.
SINTJ1: CJNE A,#02H,SINTJ2 ; TRANSMIT-DISCONNECT PENDING?
      JMP SITDIS ; YES.
SINTJ2: JMP SITRA ; TRANSMIT COMMAND IS PENDING

; ** TRANSMISSION IS DISABLED
SITDIS: JB RTS,SINTJ3 ; REQUEST TO SEND ENABLED?
      JNB TBF,SINTJ3 ; YES. TRANSMISSION DISABLED?
      ; YES.
      CLR P1.2 ; UPDATE STATUS WITH
      SETB P1.3 ; TRANSMISSION-DISABLED EVENT
      SETB P1.4 ; (STATUS=D9H)

      CLR IEO ; IGNORE PENDING EXT INTO
      CLR P1.0
      SETB P1.0 ; INTERRUPT THE 80186
      JB P3.2,$ ; WAIT TILL INTERRUPT IS ACKNOWLEDGED
      RETI

; ** A FRAME IS TRANSMITTED
SITRA: JB RTS,SINTJ3 ; A FRAME TRANSMITTED?
      ; YES.
      CLR P1.2 ; UPDATE STATUS WITH
      SETB P1.3 ; TRANSMIT-DONE EVENT
      SETB P1.4 ; (STATUS=C9).

      CLR IEO
      CLR P1.0
      SETB P1.0 ; INTERRUPT THE 80186
      JB P3.2,$ ; WAIT TILL INTERRUPT IS ACKNOWLEDGED
      RETI

; ** A FRAME IS RECEIVED
SIREC: JB RBE,SINTJ3 ; RECEIVE BUFFER FULL?
      JNB BOV,SINTJ4 ; YES. BUFFER OVERRUN?
      SETB P1.5 ; YES. SET THE E BIT
SINTJ4: MOV R0,RFL ; LOAD R0 WITH RECEIVE BYTE COUNT
      MOV R1,RBS ; LOAD R1 WITH RECEIVE BUFFER ADDRESS
      CLR IEO ; IGNORE PENDING EXT INTO (IF ANY)
      SETB PX0 ; EXT INTO: PRIORITY 1

      MOV A,@R1 ; MOVE FIRST BYTE INTO ACC.
      MOVX @DPTR,A ; WRITE TO THE COMMAND/DATA REG
      SETB P1.1 ; ENABLE DMA REQUEST
      INC R1 ; INC POINTER TO RECEIVE BUFFER
      JB P3.3,$ ; WAIT FOR DMA ACK.
      DJNZ R0,CINTJ7 ; LAST BYTE MOVED?
      SJMP CINTJ8 ; YES

CINTJ7: MOV A,@R1 ; LOAD RECEIVED DATA INTO ACC.
      MOVX @DPTR,A ; WRITE TO THE COMMAND/DATA REG.
      INC R1 ; INC POINTER TO RECEIVE BUFFER
      JB P3.3,$ ; WAIT TILL DMA ACK
      DJNZ R0,CINTJ7 ; LAST BYTE MOVED TO COMMAND/DATA REG?
      ; NO. DEPOSIT THE NEXT BYTE

CINTJ8: MOV A,RFL ; LOAD BYTE COUNT
      MOVX @DPTR,A ; WRITE TO THE COMMAND/DATA REG
      JB P3.3,$ ; WAIT FOR DMA ACK.
      MOV A,STAD ; LOAD STATION ADDRESS
      MOVX @DPTR,A ; WRITE TO THE COMMAND/DATA REG
      JB P3.3,$ ; WAIT FOR DMA ACK.
      MOV A,RCB ; LOAD RECEIVE CONTROL BYTE
      MOVX @DPTR,A ; WRITE TO THE COMMAND/DATA REG
      JB P3.3,$ ; WAIT FOR DMA ACK.
      CLR P1.1 ; CLEAR DMA REQUEST
      CLR PX0 ; EXTERNAL INTERRUPT: PRIORITY 0

```

231784-32

231784-33

Figure A-14. Serial Channel Interrupt Routine

231784-34

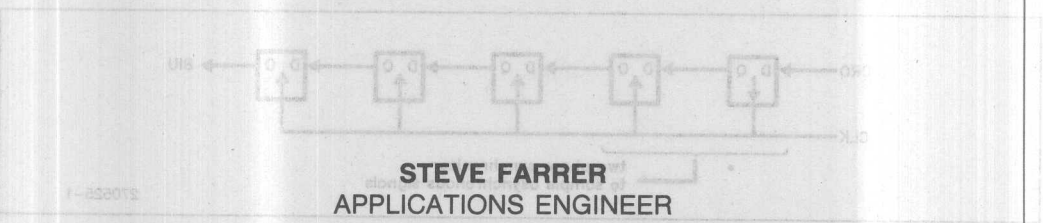
Lock conditions can also delay a DMA cycle by a significant amount, depending on the type of instruction locked. A typical locked XCHG instruction from memory to register could delay the DMA cycle by as much as 18 clocks if the memory access required two bus cycles (80188 or odd locations on the 80186). On the other hand, a locked repeat MOVSB could delay a DMA cycle by up to 1.02 million clocks depending on the number of transfers and the number of bus cycles per transfer.

Interrupt acknowledge can also affect DMA latency because the bus is locked out during the first two bus cycles required to latch the interrupt vector type. This causes the worst case latency during interrupt acknowledge to be:

4	Clocks (Minimum Setup)
10	Clocks (2 Bus Cycles + 2 Idle Clocks) Min
14	Clocks Total

Both HOLD and LOCK are extremely dependent on the type of system being designed and therefore are not really considered to be normal worst case latency. However, odd byte accesses and effective address calculations are conditions that frequently occur in almost all systems. Under these conditions of no HOLD, no LOCK, and no wait states, the worst case occurs when the DMA request occurs in an instruction data cycle.

Effective addresses (EA) always require 4 clocks for calculation and can only take place during T3-T4-T5-T6 or T7-T8-T9-T10. The EA requires an immediate value in the prefetch queue, then a signal goes active which places the EA bus cycle as a higher priority than any other BIU request. This is so the execution unit won't be waiting on the bus interface unit. If the EA hadn't required the value in the queue, then the BIU could proceed with the next instruction shortly after it had sent the request to the BIU. Figure 2 shows the effect EA calculations have on DMA latency.



When using the DMA controller of the 80186 and 80188 there are several operating conditions which affect the service time (latency) between when the DMA request is generated and when the bus cycles associated to the DMA transfer are actually run. This application first describes those conditions which affect DMA latency.

DMA REQUEST GENERATION

The minimum DMA latency is 4 clocks and, depending on when the signal arrives (i.e. if the signal just missed the setup time), it might appear to be almost 5 clocks. This 4 to 5 clock delay is due to a two phase synchronous gate delays the DRQ signal.

December 1987

If the Bus Interface Unit (BIU) is available when the DRQ signal reaches it, then a DMA cycle will proceed at 1T of the bus cycle as the next clock.

A note that the DRQ signal is not latched, and must remain active until serviced. If the DRQ signal is brought low after being asserted high, then a '0' will propagate through and if the request had not yet been serviced then the BIU will see a '0' and the cycle will never take place.

Conditions that affect DMA latency in order of worst case are as follows:

1. HOLD
 2. LOCK - INIT
 3. Odd byte accesses
 4. Effective Address Calculations (EA)
- HOLD can indirectly delay a DMA cycle. There is no mechanism internally to remove HLD when a DMA request is pending.

When using the DMA controller of the 80186 and 80188, there are several operating conditions which affect the service time (latency) between when the DMA request is generated and when the bus cycles associated to the DMA transfer are actually run. This application brief describes those conditions which affect DMA Latency.

DMA REQUEST GENERATION

The minimum DMA latency is 4 clocks and, depending on when the signal arrives (i.e. if the signal just missed the setup time), it might appear to be almost 5 clocks. This 4 to 5 clock delay is due to a two phase synchronizer and various transfer gate delays the DRQ signal must take before reaching the BIU. Conceptually the circuit looks like Figure 1.

If the Bus Interface Unit (BIU) is available when the DRQ signal reaches it, then a DMA cycle will proceed at T1 of the bus cycle as the next clock.

Also note that the DRQ signal is not latched, and must remain active until serviced. If the DRQ signal is brought low after being asserted high, then a '0' will propagate through and; if the request had not yet been serviced, then the BIU will see a '0' and the cycle will never take place.

Conditions Affecting DMA Latency

The circumstances that affect DMA latency in order of worst case are as follows:

- 1) HOLD
- 2) LOCK - INTA
- 3) Odd byte accesses
- 4) Effective Address Calculations (EA)

HOLD can indefinitely delay a DMA cycle. There is no mechanism internally to remove HLDA when a DMA request is pending.

LOCKed instructions can also delay a DMA cycle by a significant amount, depending on the type of instruction locked. A typical locked XCHG instruction from memory to register could delay the DMA cycle by as much as 18 clocks if the memory access required two bus cycles (80188 or odd locations on the 80186). On the other hand, a locked repeat MOVS could delay a DMA cycle by up to 1.05 million clocks depending on the number of transfers and the number of bus cycles per transfers.

Interrupt acknowledges can also affect DMA latency because the bus is locked out during the first two bus cycles required to fetch the interrupt vector type. This causes the worst case latency during interrupt acknowledges to be:

4	Clocks (Minimum Setup)
10	Clocks (2 Bus Cycles + 2 Idle Clocks) Min
14	Clocks Total

Both HOLD and LOCK are extremely dependent on the type of system being designed and therefore are not really considered to be normal worst case latency. However, odd byte accesses and effective address calculations are conditions that frequently occur in almost all systems. Under these conditions of no HOLD, no LOCK, and no wait states, the worst case occurs when the DMA request loses to an instruction data cycle requiring an effective address calculation.

Effective addresses (EA) always require 4 clocks for calculation and can only take place during T3-T4-TI-TI, T4-TI-TI-TI, or TI-TI-TI-TI. This creates an extra minimum insertion of 2 T-idle cycles. If the EA requires an immediate value in the prefetch queue, then a signal goes active which places the EA bus cycle at a higher priority than any other BIU requests. This is so the execution unit won't be waiting on the bus interface unit. If the EA hadn't required the value in the queue, then the EU could proceed with the next instruction shortly after it had sent the request to the BIU. Figure 2 shows the effects EA calculations have on DMA Latency.

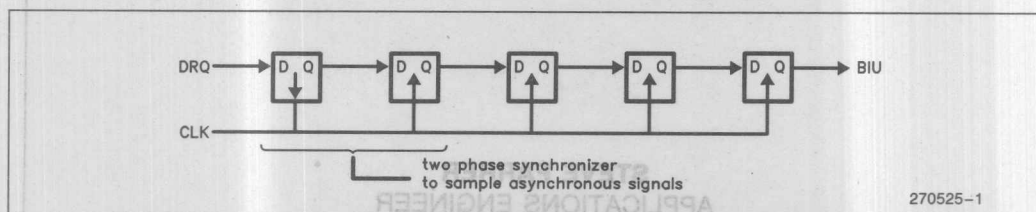


Figure 1. DMA Request Synchronization

Address	Code	Instruction
FA058	90	NOP
FA059	90	NOP
FA05A	2E87060100	XCHG AX, CS:WORD PTR 0001

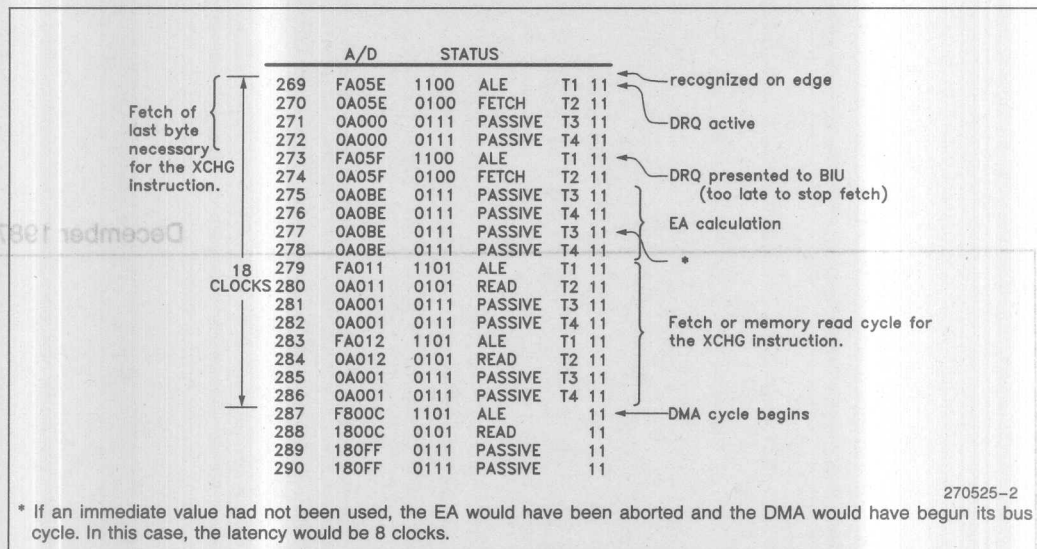
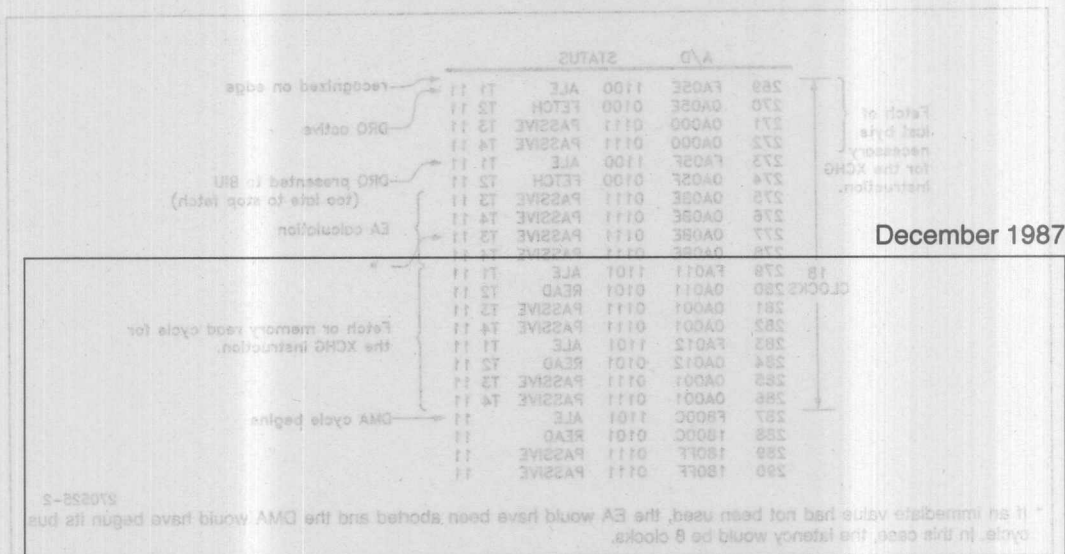


Figure 2. Logic State Analyzer Trace and Accompanying Program Code



80186/80188 EFI Drive and Oscillator Operation

STEVE FARRER
APPLICATIONS ENGINEER

There has been some confusion in the past regarding the correct input for EFI (External Frequency Input) use and what parameters should be used for crystal selection. This Application Brief discusses the trade-offs with each input so that one can decide which input suits his design and also lists the parameters for crystal selection.

EFI Operation

The oscillator circuit on the 186/188 is as shown in Figure 1 (simplified). Either input may be used for an EFI signal. Using X1 requires very little drive from an external oscillator since it is essentially the gate of an NMOS transistor. Clock operation works fine using this input, but at higher frequencies the stray capacitance on X2 begins to change the duty cycle of the clock. This will eventually cause the part to fail.

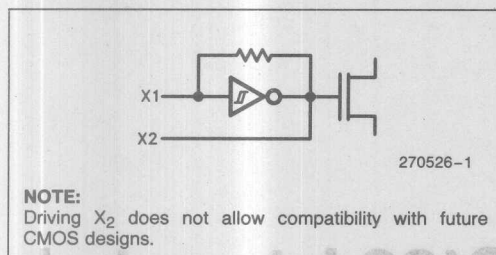


Figure 1. Oscillator Circuit on the 186/188

Using X2 as an EFI gives a broader frequency range but places a more stringent requirement on the drive

capability of the external oscillator. Since X1 is an input, it may be grounded to minimize the capacitance. This in turn allows for a higher frequency range since the duty cycle remains closer to 50%. But with X1 grounded, the output of the inverter (which is directly connected to X2) is always trying to output a high. This means the oscillator driving X2 must be capable of sinking up to 15 mA at cold temperatures when trying to drive it low. If the external oscillator is capable of supplying 15 mA, then this method is preferred. Otherwise, X1 should be used as an EFI.

Caution: using X2 for EFI does not allow for CMOS compatibility at a future date.

Crystal Operation

The oscillator circuit is a single stage amplifier connected as a Pierce oscillator. There are no passive components in the oscillator circuit, only a unique combination of depletion and enhancement mode FET's. Characterization of the oscillator circuit showed that operation was optimum with crystal parameters as follows:

ESR (Equivalent Series Resistance)	30 ohms maximum
Co (Shunt Capacitance)	7.0 pf max.
C1 (Load Capacity)	20 pf \pm 2 pf
Drive Level	1 mW max.

This characterization data was supplied by:

Standard Crystal Corporation
9940 East Baldwin Place
El Monte, CA 91731
(213) 443-2121

The 80C186/80C188 Integrated Refresh Control Unit

© Intel Corporation, 1987

The 80C186 and 80C188 incorporate a special control unit that integrates address and clock counters which, along with the Bus Interface Unit (BIU), facilitates dynamic memory refreshing. Refreshing is an operation required by dynamic memory to ensure data retention.

Dynamic memory refreshing can be controlled using anything from an exotic memory controller to a simple timer along with a DMA controller. In fact, the 80C186 device accomplishes the task memory refreshing by using one of the internal timer/counters and a DMA channel. However, doing this meant that very desirable internal functions were no longer available to do more useful work.

Dynamic memory, unlike static or non-volatile memory, always require some form of a memory controller to enable read and write operations. Therefore, even the most basic dynamic memory interface has a minimum set of support logic. The advent of programmable logic and highly integrated dynamic memory has made the job of designing a memory controller somewhat straightforward. However, directly supporting memory refresh still can complicate many controller designs.

The designer of a memory controller must take into account CPU-versus-refresh arbitration and must provide a mechanism to generate periodic refresh requests. Most dynamic memory devices now contain internal

refresh address counters which eliminate the need for external refresh address generation. However, such devices tend to complicate a memory controller design. The 80C186 simplifies dynamic memory controller design by integrating a refresh mechanism into the operation of the CPU.

This application brief is not intended to be a discussion of dynamic memory controller design. Instead, it will concentrate on the operation of the Refresh Control Unit with the 80C186, and how it can help simplify a memory controller.

The discussions on the following pages apply to BOTH the 80C186 and 80C188 except where noted.

UNDERSTANDING DYNAMIC MEMORY

Before explaining how memory refreshing is accomplished, some understanding of a Dynamic Random Access Memory (DRAM) device is needed. Figure 1 shows a simplified block diagram of a DRAM device, while a block diagram of a typical dynamic memory controller is shown in Figure 2.

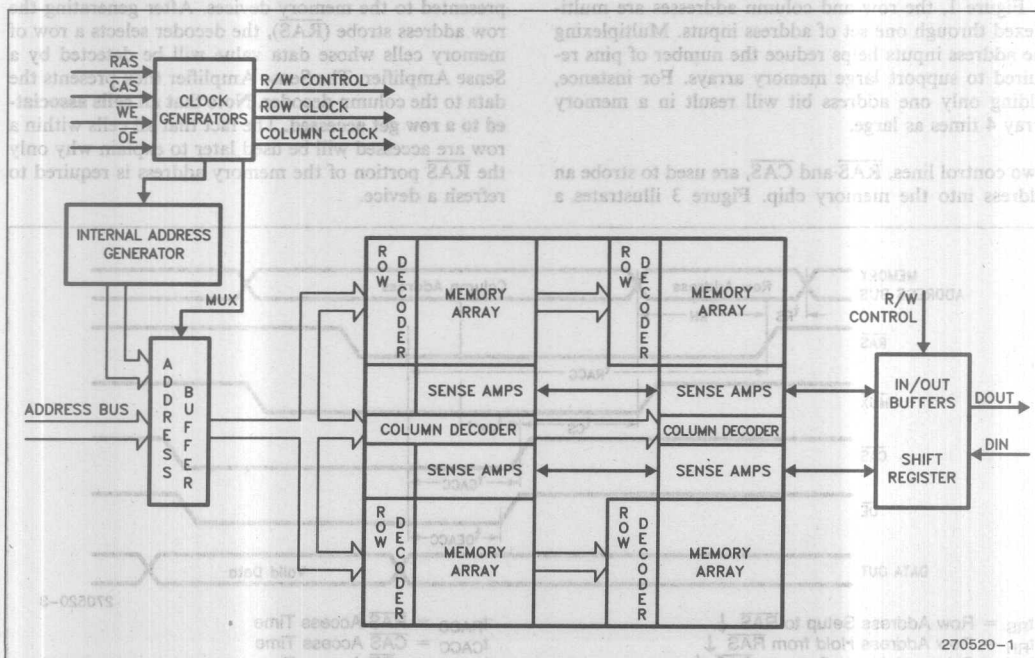


Figure 1. Random Access Memory Device

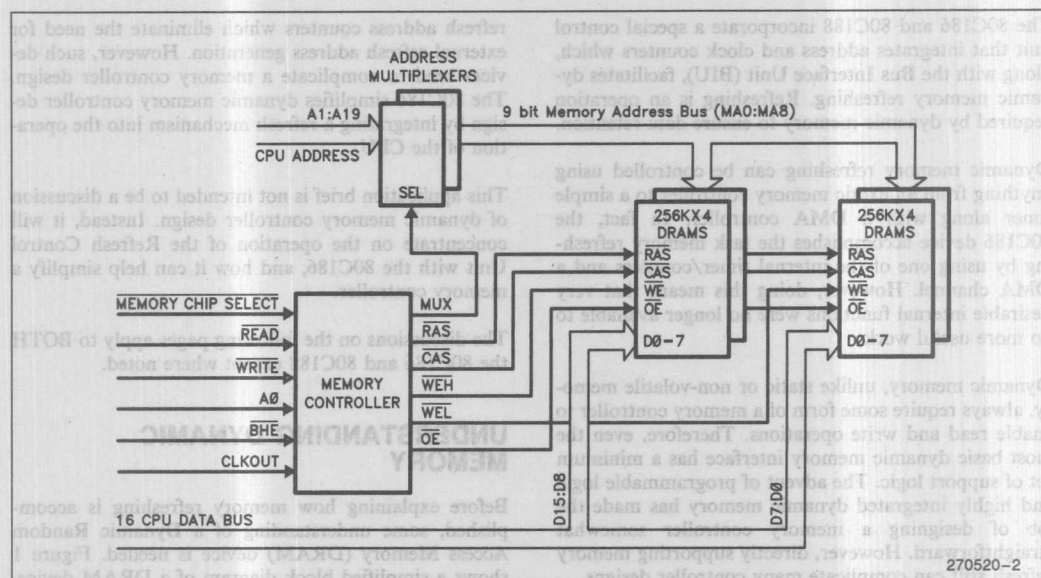


Figure 2. Minimum Configuration Memory Controller

The typical DRAM memory array is built as a matrix. Thus, any bit or cell in the memory array is accessed by specifying a unique row and column address. As shown in Figure 1, the row and column addresses are multiplexed through one set of address inputs. Multiplexing the address inputs helps reduce the number of pins required to support large memory arrays. For instance, adding only one address bit will result in a memory array 4 times as large.

Two control lines, $\overline{\text{RAS}}$ and $\overline{\text{CAS}}$, are used to strobe an address into the memory chip. Figure 3 illustrates a

timing diagram for a typical memory read access and the relationship between the $\overline{\text{RAS}}$ and $\overline{\text{CAS}}$ signals. The signal $\overline{\text{MUX}}$ controls which half of the address is presented to the memory devices. After generating the row address strobe ($\overline{\text{RAS}}$), the decoder selects a row of memory cells whose data value will be detected by a Sense Amplifier. The Sense Amplifier then presents the data to the column decoder. Note that all cells associated to a row get accessed. The fact that all cells within a row are accessed will be used later to explain why only the $\overline{\text{RAS}}$ portion of the memory address is required to refresh a device.

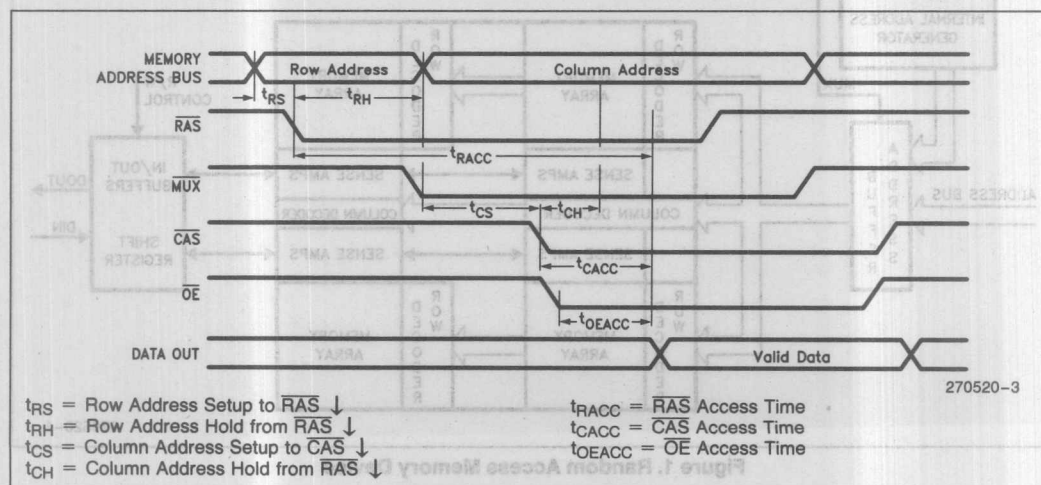


Figure 3. DRAM Signal Timings

When the column address is strobed, it is here that only one of the memory cells is selected. The memory cell will either be written to or read from depending on the the control signals \overline{WE} and \overline{OE} respectively. Since data from one entire row is presented to the column decoder, it is possible (on some devices) to simply cycle through column addresses to access additional data. The basic idea, however, is that two sets of addresses are required to access a memory cell within a memory array. Furthermore, specifying a single row address internally accesses all memory cells within that row.

The minimum memory controller interface consists of a sequencer and an address multiplexer. The sequencer is responsible for generating the correct control signals: \overline{RAS} ; \overline{CAS} ; \overline{MUX} ; \overline{WE} ; \overline{OE} . The address multiplexer logic is responsible for translating the processor address bus to the memory address bus. These two pieces of logic can exist in any form, from simple TTL gates to single chip solutions. However, what is missing from the simplified memory controller is a mechanism to perform memory refresh.

UNDERSTANDING MEMORY REFRESH

As indicated earlier, dynamic memory needs to be refreshed in order to maintain its data. Refreshing is accomplished whenever a memory cell is accessed. It is not necessary to read a memory location and then write the value back in order to refresh a memory cell. Simply cycling through a complete set of row addresses is all that is required. Remember, since a row accesses all memory cells associated to it, accessing all rows will access all the cells within the device.

Referring back to Figure 2, the 9 address bits presented to the memory devices are multiplexed from the 18 bits of address generated by the 80C186. In the design, address bits A1-A9 are presented during \overline{RAS} , while address bits A10-A18 are presented during \overline{CAS} . Note that address bit A0 is not used because the memory array is organized as word wide; A0 along with BHE are used to select one or both of the bytes within a word.

Cycling through row addresses is the only requirement needed to refresh a DRAM device. Using the example in Figure 2, 9 bits of address are needed. Nine bits represent 512 unique addresses, and the only requirement is that each unique address be regenerated every

8 ms (maximum refresh rate for most devices with 512 rows). An 8 ms refresh interval divided by 512 addresses results in an average refresh cycle rate of 15.625 microseconds. Therefore, every 15.625 microseconds a mechanism must exist that will access the DRAM device, each time presenting a new row address. Any rate faster than 15.625 microseconds is acceptable, but significantly faster times have the potential of decreasing memory performance.

WAYS TO REFRESH A MEMORY DEVICE

For most dynamic memory devices, there are several ways in which a refresh cycle can be run. The first and simplest way is to generate memory read cycles every 15.6 microseconds. Each new memory read cycle would generate a unique address. When refreshing is accomplished using memory read cycles, the memory controller is simplified. Only the basic control signals need to be generated, which are the minimum needed to access the memory anyway. Simplicity is, however, accompanied by one drawback; bus overhead. Using memory reads to perform DRAM refreshing means that one bus cycle every 15.6 microseconds is wasted. When operating at very slow speeds, a wasted bus cycle might appear to be significant. But if a bus cycle takes only, say, 320 nanoseconds to complete, running a refresh cycle every 15.6 microseconds represents a two percent hit in bus performance.

A second method relies on the fact that most dynamic memory devices now have built in refresh address mechanisms. DRAM refreshing can be accomplished by generating \overline{CAS} before \overline{RAS} signaling (see Figure 4a). This method requires that an external signal generate a periodic request to the DRAM controller to initiate the refresh cycle. A method similar to \overline{CAS} before \overline{RAS} refreshing is hidden refresh. Figure 4c illustrates the timing involved to perform hidden refresh. No request logic is needed, since the memory access itself is what initiates the refresh cycle. However, constant memory accessing is required in order to maintain refreshing. Once accessing stops, refreshing stops. Both of the methods described have the advantage of not consuming bus bandwidth, but require the memory controller to handle the somewhat different (from normal memory accessing) signaling requirements.

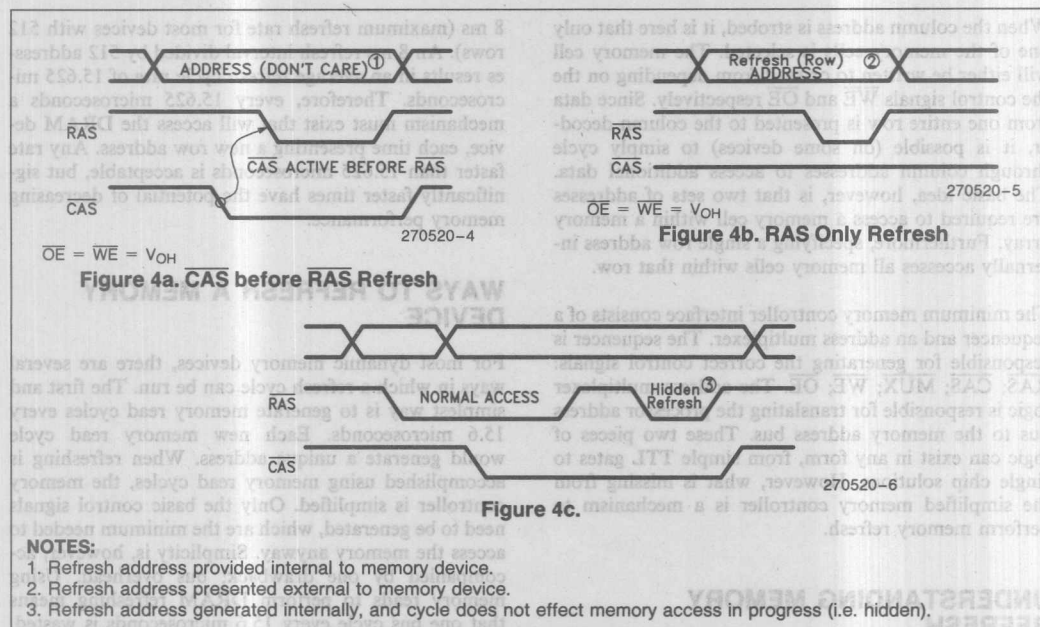


Figure 4. Alternate Refreshing Methods

A final method is to implement a discrete design that supports refresh control and refresh address generation. The circuit details are shown in Figure 4b. A discrete design allows the most design flexibility and can be tailored to meet any system-to-memory interfacing requirements.

There are other methods available, most of which involve single-chip dedicated memory controllers. However, any memory controller design that performs the function of refreshing either directly or through external support circuitry has one major concern; arbitration between the refresh cycle and a normal memory access. The best way to make the operation of the DRAM memory controller a true slave to the operation of the CPU is to include refreshing as part of the functionality of the CPU. By offloading the task of memory refreshing onto the CPU, the memory controller can be simplified and dedicated to the duty of DRAM interfacing.

The idea that the 80C186 refresh cycle is simply a memory read means that the dynamic memory control logic does not need to differentiate between refresh cycles and normal memory read cycles. This simplifies the design of the memory controller. There are no special signaling requirements needed, and RAS only refreshing (for low-power designs) can be easily accommodated. Further, since the request is generated internally and synchronous with the operation of the BIU, no special external logic needs to detect when a refresh cycle conflicts with a CPU access.

Cycling through row addresses is the only requirement needed to refresh a DRAM device. Using the example in Figure 2, 9 bits of address are needed. Nine bits represent 512 unique addresses, and the only requirement is that each unique address be regenerated every

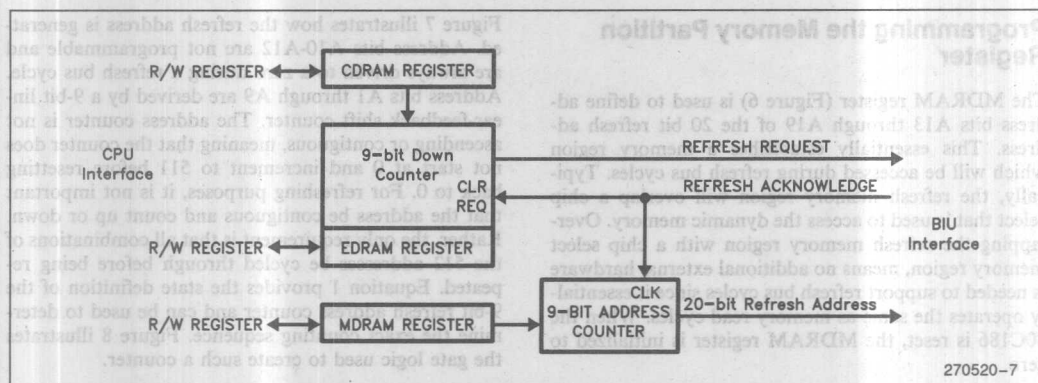


Figure 5. Refresh Control Unit Block Diagram

80C186 REFRESH CONTROL FEATURES

The Refresh Control Unit (RCU) of the 80C186 consists of a 9-bit address counter, a 9-bit down counter, and support logic. The block diagram can be seen in Figure 5.

The 9-bit address counter is controlled by the BIU and used whenever a refresh bus cycle is executed. Thus, any dynamic memory device whose refresh address requirement does not exceed nine bits can be directly supported by the 80C186. A special register has been defined to allow the base (starting) address of the refresh memory region to be specified. This base address can be located on any 4 kilobyte boundary. Furthermore, if this refresh base address overlaps any of the defined chip select regions, the chip select defined for that region will go active.

The 9-bit down counter initiates a refresh event. When the counter decrements to 1 (it decrements every clock cycle), a refresh request is presented to the BIU. When the bus is free, the BIU will run the refresh (memory) bus cycle. Note that since a refresh bus cycle is executed by the BIU, the faster refresh cycles are requested the greater the impact on bus performance. Referring back to the discussion of request rates, the maximum refresh period is typically 15.6 microseconds. With the 80C186 operating at 12.5 MHz, this represents a refresh bus impact of only 2%. However, at 5 microseconds the bus impact is 15%. Therefore, the refresh request rate should be tailored to meet the needs of the dynamic memory and the system. The 80C186 provides flexibility by allowing the request rate to be programmable in 80 ns steps (at 12.5 MHz).

To facilitate low power designs, the refresh bus cycle provides a mechanism whereby the dynamic memory

devices can be turned off during refresh accesses. Low power control is accomplished by driving both address bit A0 and the control signal \overline{BHE} to a high level. Essentially an invalid bus access condition exists, since A0 and \overline{BHE} are used to indicate which half of the data bus is being accessed. When both are high during the access, the indication is that neither half of the bus is being used for the data transfer. This is acceptable for refresh bus cycles since no data is actually being transferred. If the memory controller takes advantage of this condition, the output enables of the dynamic memory devices (as well as the \overline{CAS} strobe) can be disabled during refresh bus cycles, providing overall lower power consumption.

PROGRAMMING CHARACTERISTICS OF THE REFRESH CONTROL UNIT

A block of control registers are defined in the Peripheral Control Block (PCB) that define the operating characteristics of the refresh control unit (refer to Figure 5). These registers are only accessible when the 80C186 is operating in enhanced mode. When in compatibility mode, the 80C186 will ignore any reads or writes to the RCU registers.

The three registers associated with the refresh unit (MDRAM, CDRAM, EDRAM) provide the following features:

- 1) Enable/disable refresh unit
- 2) Establish a refresh request rate
- 3) Establish a refresh memory region
- 4) Examine the refresh down counter

It is not necessary to program any of these registers in a specific sequence, although the refresh request rate and refresh base address registers should be programmed before the refresh unit is enabled.

Programming the Memory Partition Register

The MDRAM register (Figure 6) is used to define address bits A13 through A19 of the 20 bit refresh address. This essentially establishes a memory region which will be accessed during refresh bus cycles. Typically, the refresh memory region will overlap a chip select that is used to access the dynamic memory. Overlapping the refresh memory region with a chip select memory region, means no additional external hardware is needed to support refresh bus cycles since it essentially operates the same as memory read cycles. When the 80C186 is reset, the MDRAM register is initialized to zero.

Figure 7 illustrates how the refresh address is generated. Address bits A10-A12 are not programmable and are always driven to a zero during a refresh bus cycle. Address bits A1 through A9 are derived by a 9-bit linear-feedback shift counter. The address counter is not ascending or contiguous, meaning that the counter does not start at 0 and increment to 511 before resetting back to 0. For refreshing purposes, it is not important that the address be contiguous and count up or down. Rather, the only requirement is that all combinations of the 512 addresses be cycled through before being repeated. Equation 1 provides the state definition of the 9-bit refresh address counter and can be used to determine the exact counting sequence. Figure 8 illustrates the gate logic used to create such a counter.

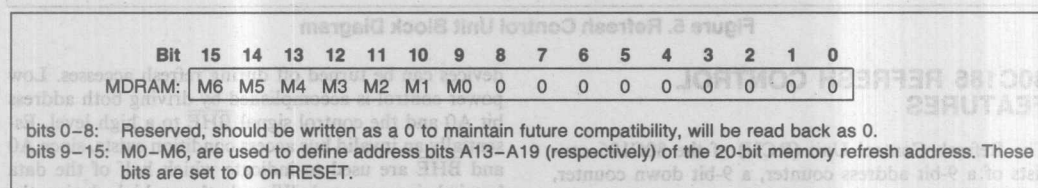


Figure 6. MDRAM Register Format

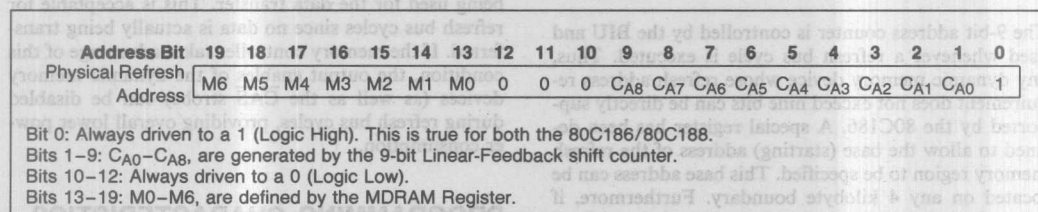
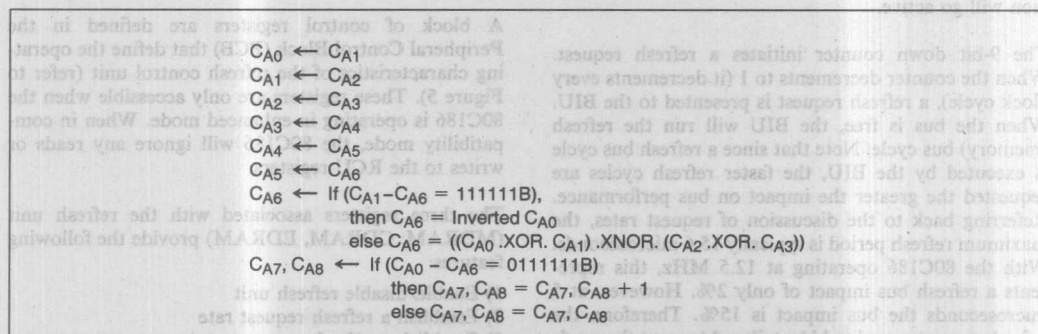


Figure 7. Physical Refresh Address Generation



Equation 1. Refresh Counter Operation

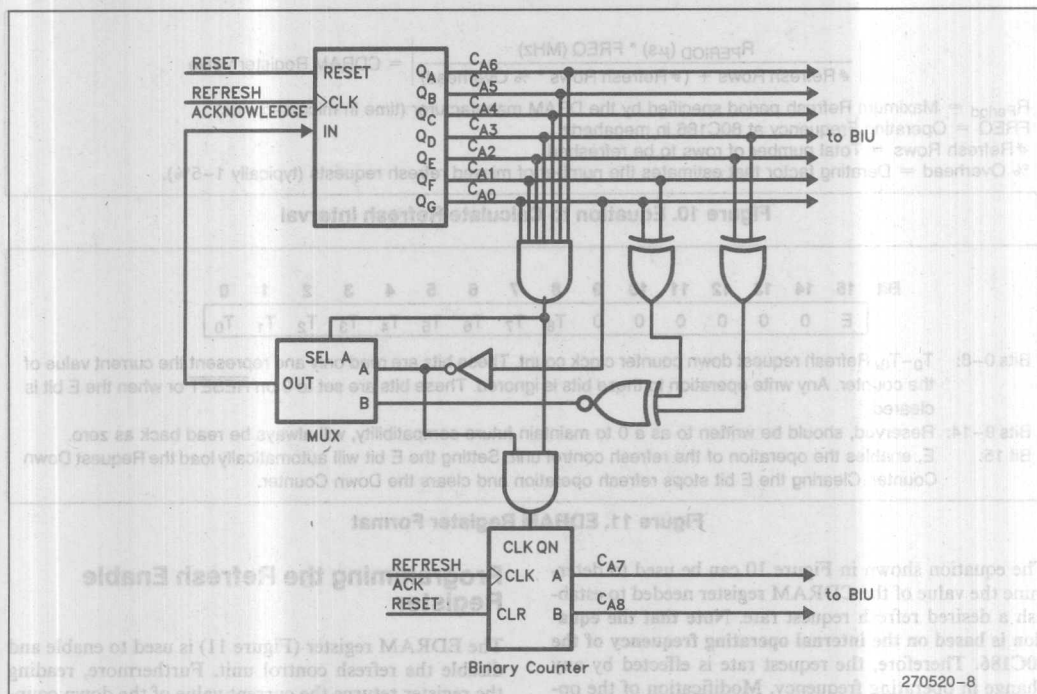


Figure 8. Logic Representation of Refresh Address Counter

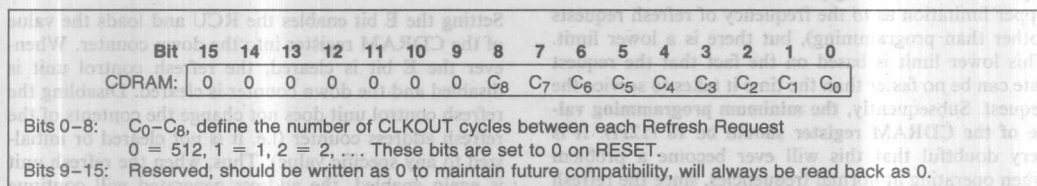


Figure 9. CDRAM Register Format

There are no limitations placed on the programming of the MDRAM register, but be aware that any chip select memory region that overlaps the address established by the MDRAM register will be activated during refresh bus cycles. Therefore, the register should be programmed to correspond to the chip select address that is activated for the dynamic memory partition.

Programming the Refresh Clock Register

The CDRAM register (Figure 9) is used to define the rate at which refresh requests will be internally generated. The CDRAM register is used to maintain the start-

ing value of a down counter, which decrements each falling edge of CLKOUT. When the counter decrements to 1, a refresh request is generated and the counter is again loaded with the value contained in the CDRAM register. Initially, however, the contents of the CDRAM register is loaded into the down counter when the enable bit in the EDRAM register set. Thus, if the CDRAM register is changed, the new value will take effect when either the down counter reaches 1 and reloads itself, or whenever the E bit is written to a 1 (this is true whether the bit was previously set or not). When the 80C186 is reset, the CDRAM register is initialized to zero. A value of zero in the CDRAM register is used to indicate the maximum count rate of 512 clocks.

$$\frac{R_{PERIOD} (\mu s) * FREQ (MHz)}{\# Refresh Rows + (\# Refresh Rows * \% Overhead)} = CDRAM Register Value$$

R_{Period} = Maximum Refresh period specified by the DRAM manufacturer (time in microseconds).
 $FREQ$ = Operating Frequency at 80C186 in megahertz.
 $\# Refresh Rows$ = Total number of rows to be refreshed.
 $\% Overhead$ = Derating factor that estimates the number of missed refresh requests (typically 1–5%).

Figure 10. Equation to Calculate Refresh Interval

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	E	0	0	0	0	0	0	T_8	T_7	T_6	T_5	T_4	T_3	T_2	T_1	T_0

Bits 0–8: T_0 – T_8 , Refresh request down counter clock count. These bits are read only and represent the current value of the counter. Any write operation to these bits is ignored. These bits are set to 0 on \overline{RESET} or when the E bit is cleared.
 Bits 9–14: Reserved, should be written to as a 0 to maintain future compatibility, will always be read back as zero.
 Bit 15: E, enables the operation of the refresh control unit. Setting the E bit will automatically load the Request Down Counter. Clearing the E bit stops refresh operation and clears the Down Counter.

Figure 11. EDRAM Register Format

The equation shown in Figure 10 can be used to determine the value of the CDRAM register needed to establish a desired refresh request rate. Note that the equation is based on the internal operating frequency of the 80C186. Therefore, the request rate is effected by any change in operating frequency. Modification of the operating frequency can occur in two ways: modifying the input clock or entering power-save mode. There is no upper limitation as to the frequency of refresh requests (other than programming), but there is a lower limit. This lower limit is based on the fact that the request rate can be no faster than the time it takes to service the request. Subsequently, the minimum programming value of the CDRAM register should be 18 (12H). It is very doubtful that this will ever become a problem when operating at normal frequencies, since the refresh rate of most dynamic memories is well above this minimum programming value.

However, when making use of the power-save feature of the 80C186, it is possible to lower the operating frequency such that it will prevent adequate refreshing rates. When operating at 12.5 MHz, dividing the clock by 16 results in a cycle time of 1.28 microseconds. Since the minimum value of the CDRAM is 18, the minimum refresh rate is 23.04 microseconds. 23 microseconds is not fast enough to service most dynamic memories. Therefore, caution must be exercised when using the power-save feature of the 80C186. When there is a need to keep dynamic memory alive, the clock should not be divided much below 2 MHz to avoid monopolizing the bus with refresh activity. If there is no desire to keep memory alive during power-save operation, then the refresh unit can simply be disabled during this time.

Programming the Refresh Enable Register

The EDRAM register (Figure 11) is used to enable and disable the refresh control unit. Furthermore, reading the register returns the current value of the down counter.

Setting the E bit enables the RCU and loads the value of the CDRAM register into the down counter. Whenever the E bit is cleared, the refresh control unit is disabled and the down counter is cleared. Disabling the refresh control unit does not change the contents of the refresh address counter (i.e. it is not cleared or initialized to any specific value). Thus, when the refresh unit is again enabled, the address generated will continue from where it left off. Resetting the 80C186 automatically clears the E bit. There are no refresh bus cycles during a reset.

The current value of the down counter, as well as the present state of the E bit can be examined whenever the EDRAM register is read. Any unused bits will be returned as zero. Whenever the E bit is cleared, the T_0 through T_8 bits will be read as zero.

REFRESH CONTROL UNIT OPERATION

Figure 12 illustrates the two major operational functions of the refresh control unit that are responsible for initiating and controlling DRAM refresh bus cycles.

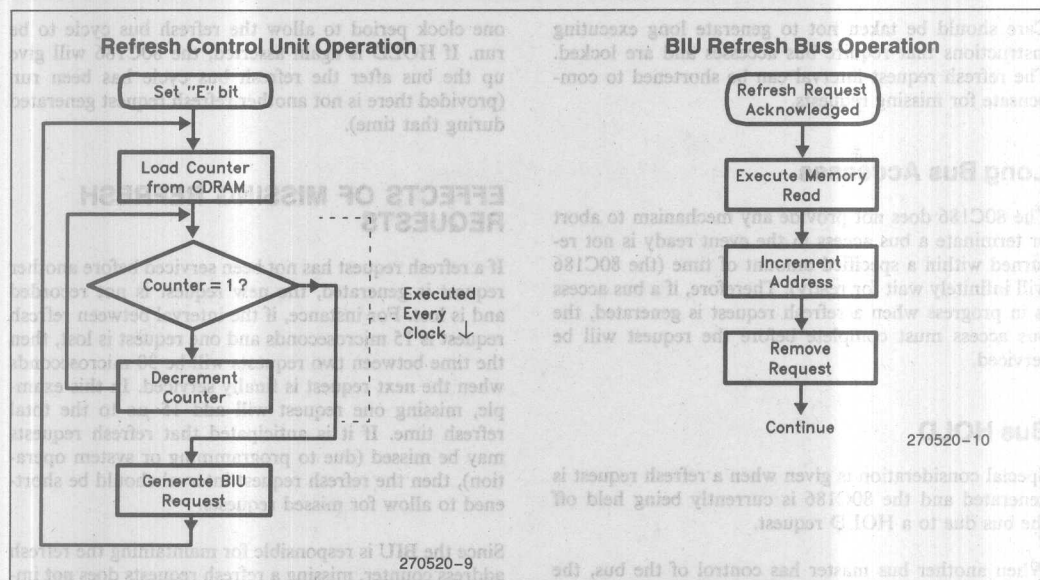


Figure 12. Flowchart of RCU Operation

The down counter is loaded (with the contents of the CDRAM register) on the falling edge of CLKOUT, either when the EFRSH bit is set or whenever the counter decrements to 1. Once loaded, the down counter will decrement every falling edge of CLKOUT. It will continue to decrement as long as the EFRSH bit remains set.

When the down counter finally decrements to 1, two things will happen. First, a request is generated to the BIU to run a refresh bus cycle. The request remains active until the bus cycle is run. Second, the down counter is reloaded with the value contained in the CDRAM register. At this time, the down counter will again begin counting down every clock cycle, it does not wait until the request has been serviced. This is done to ensure that each refresh request occurs at the correct interval. Otherwise, if the down counter only started after the previous request were service, the time between refresh requests would also be a function of bus activity, which for the most part is unpredictable. When the BIU services the refresh request, it will clear the request and increment the refresh address.

80C188 Address Considerations

The physical address that is generated during a refresh bus cycle is shown in Figure 7, and it applies to both the 80C186 and 80C188. For the 80C188, this means that the lower address bit A0 will not toggle during refresh operation. Since the 80C188 has an 8-bit external bus, A0 is used as part of memory address decod-

ing. Whereas the 80C186, with its 16-bit external bus, uses A0 (along with BHE) to select memory banks. Therefore, when designing 80C188 memory subsystems it is important not to include A0 as part of the ROW address that is used as a refresh address. Appendix A illustrates Memory Address Multiplexing Techniques that can be applied to the 80C186 and the 80C188.

MISSING REFRESH REQUESTS

Under most operating conditions, the frequency of refresh requests is a small percentage of the bus bandwidth. Still, there are several conditions that may prevent a refresh request from being serviced before another request is generated. These conditions include:

- 1) LOCKED Bus Cycles
- 2) Long Bus accesses (wait states)
- 3) Bus HOLD

LOCKED Bus Cycles

Whenever the bus is LOCKED, the CPU maintains control of the BIU and will not relinquish it until the locked operation is complete. Therefore, internal operations like refresh and DMA are not allowed to execute until the LOCKED instruction has completed. Where this presents the greatest problem is when an instruction such as a move string is executed, and is locked. The move string instruction can take from several clocks to hundreds of thousands of clocks to complete. Obviously anything that takes longer than 512 clocks to complete will always cause a refresh overflow.

Care should be taken not to generate long executing instructions that require bus accesses and are locked. The refresh request interval can be shortened to compensate for missing requests.

Long Bus Accesses

The 80C186 does not provide any mechanism to abort or terminate a bus access in the event ready is not returned within a specified amount of time (the 80C186 will infinitely wait for ready). Therefore, if a bus access is in progress when a refresh request is generated, the bus access must complete before the request will be serviced.

Bus HOLD

Special consideration is given when a refresh request is generated and the 80C186 is currently being held off the bus due to a HOLD request.

When another bus master has control of the bus, the HLDA signal is kept active as long as the HOLD input remains active. If a refresh request is generated while HOLD is active, the 80C186 will remove (drive inactive) the HLDA signal to indicate to the other bus master that the 80C186 wishes to regain control of the bus (see Figure 13). If, and only if, the HOLD input is removed will the BIU begin to run the refresh bus cycle.

Therefore, it is the responsibility of the system designer to ensure that the 80C186 can regain the bus if a refresh request is signaled. The sequence of HLDA going inactive while HOLD is active can be used to signal a pending refresh request. HOLD need only go inactive for

one clock period to allow the refresh bus cycle to be run. If HOLD is again asserted, the 80C186 will give up the bus after the refresh bus cycle has been run (provided there is not another refresh request generated during that time).

EFFECTS OF MISSING REFRESH REQUESTS

If a refresh request has not been serviced before another request is generated, the new request is not recorded and is lost. For instance, if the interval between refresh requests is 15 microseconds and one request is lost, then the time between two requests will be 30 microseconds when the next request is finally serviced. In this example, missing one request will add 15 μ s to the total refresh time. If it is anticipated that refresh requests may be missed (due to programming or system operation), then the refresh request interval should be shortened to allow for missed requests.

Since the BIU is responsible for maintaining the refresh address counter, missing a refresh requests does not imply that refresh addresses are skipped. In fact, an address can never be skipped unless a reset occurs.

CONCLUSION

The Internal Refresh Control Unit of the 80C186 and 80C188 helps solve three issues concerning DRAM refreshing: a way to generate periodic refresh requests; a way to generate refresh addresses; a way to simplify DRAM memory controllers. Once a memory controller has been designed to handle the simple tasks of reading and writing the task of refreshing has already been built in.

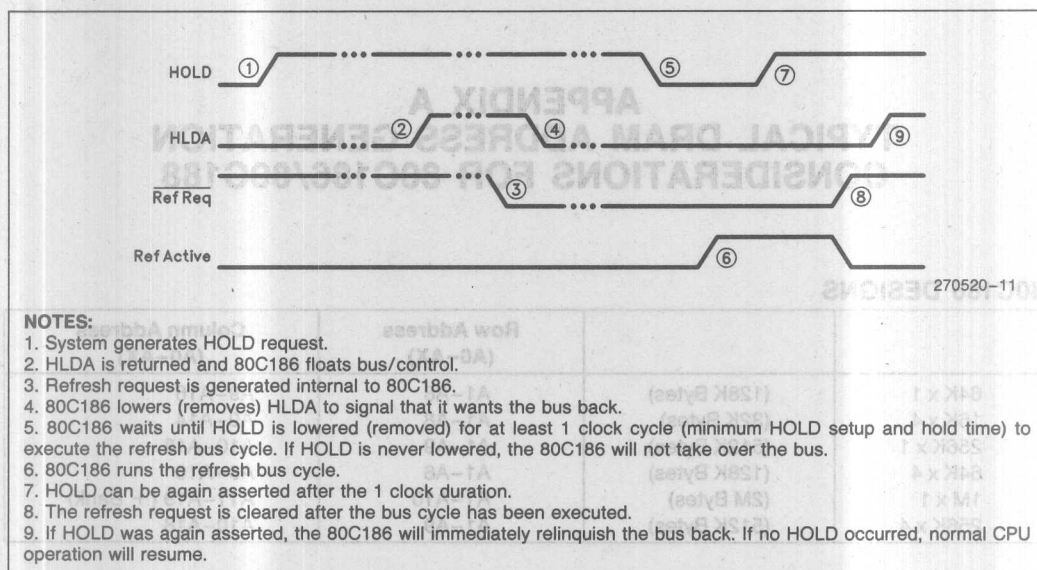


Figure 13. HOLD/HLDA Timing and Refresh Request

APPENDIX A TYPICAL DRAM ADDRESS GENERATION CONSIDERATIONS FOR 80C186/80C188

80C186 DESIGNS

		Row Address (A0-AX)	Column Address (A0-AX)
64K x 1	(128K Bytes)	A1-A8	A9-A16
16K x 4	(32K Bytes)	A1-A8	A9-A14
256K x 1	(512K Bytes)	A1-A9	A10-A18
64K x 4	(128K Bytes)	A1-A8	A9-A16
1M x 1	(2M Bytes)	A1-A10	A11-A19 (+ Bank)
256K x 4	(512K Bytes)	A1-A9	A10-A18

80C188 DESIGNS

NOTE:

Address bit A0 can be used in either RAS or CAS addresses, so long as it is not included in any refresh address bits.

		Row Address (A0-AX)	Column Address (A0-AX)
64K x 1	(64K Bytes)	A1-A7, A0	A8-A15
16K x 4	(16K Bytes)	A1-A7, A0	A8-A13
256K x 1	(256K Bytes)	A1-A8, A0	A9-A17
64K x 4	(64K Bytes)	A1-A8	A0, A9-A15
1M x 1	(1M Byte)	A1-A9, A0	A10-A19
256K x 4	(256K Bytes)	A1-A9	A0, A10-A17

RAM Type	RAS Add	CAS Add	Refresh Add
64K x 1	A0-A7	A0-A7	A0-A6
16K x 4	A0-A7	A0-A5	A0-A6
256K x 1	A0-A8	A0-A8	A0-A7
64K x 4	A0-A7	A0-A7	A0-A7
1M x 1	A0-A9	A0-A9	A0-A8
256K x 4	A0-A8	A0-A8	A0-A8

All the chip selects so that READY goes active whenever all the memory chip selects are inactive (i.e. the cycle is not in a valid memory region).

BUS OVERHEAD

The absolute maximum overhead can be calculated at a given speed by taking the number of refresh cycles divided by the total number of bus cycles for a given period of time. At 8 MHz these values can be calculated as follows:

$$\frac{2 \text{ bus cycles}}{18.3 \text{ ns/500 ns}} \times 100 = 8.8\% \text{ maximum overhead}$$

In reality, the bus overhead associated with the DMA cycles is much lower due to the instruction prefetch queue. When a DMA cycle is requested by the timer for a refresh cycle, the Bus Interface Unit honors the request on the next bus cycle boundary (with the exception of LOCKed bus cycles and odd aligned accesses). Typically, this time is idle time on the bus and the impact on the overall performance is extremely small. The following table shows more realistic data which was acquired by running 6 different benchmarks with and without the DMA channel enabled to provide refresh every 15.3µs.

BENCHMARK OVERHEAD (%)		
5.5%	5.9%	5.5%
3.4%	3.5%	3.4%

The program which showed the highest bus overhead tended to be very bus intensive. Also note that at faster frequencies the bus overhead becomes even less.

DMA OPERATION

The DMA controller is programmed to be source synchronized with the TC (transfer count) bit cleared. This ensures that the DMA controller never reaches a final count. The source pointer continues to increment through memory on every cycle. When FFFFH is reached, the address rolls over to 0000H.

The programming values for the DMA registers are shown in Figure 1. The source pointer may be initialized to a refresh cycle falls where there is no memory. This can easily be accomplished by either decoding a couple of high order address lines or

STEVE FARRER
APPLICATIONS ENGINEER

In many low-cost 80186/80188 designs, dynamic memory offers an excellent cost/performance advantage. However, DRAM refreshing is often complicated by the need to perform memory refreshing. This application brief describes how to use the Timer and DMA functionality of the 80186/80188 to perform memory refresh.

THEORY OF OPERATION

Dynamic RAM refreshing is accomplished by strobing a ROW address to every ROW of the DRAM within a given period of time. One way to do this is to perform periodic sequential reads to the DRAM using a DMA controller. This can be achieved with the 80186/80188 using Timer 2 and one of the DMA channels such that the timer generates one DMA cycle approximately every 15 micro-seconds. Please note that this is a single row refresh method and not a burst refresh. Single row refreshing reduces the bus overhead considerably when compared to burst refreshing.

The control logic of the DRAM is such that a RAS (row address strobe) occurs on every memory read, regardless of the address. This is necessary because the DMA channel is cycling through the entire 1 MByte address space and the address of the refresh cycle does not always fall within the range of the DRAM bank.

Although the lower address bits continue to count, the lower address bits continue to provide the row address.

READY LOGIC WITH MEMORY

Since the DMA controller is cycling through the entire 1 MByte address space, care must be taken to ensure that a READY signal is available for all addresses. One way to do this is to use only the internal wait state generator for memory access and to strap the SRDY and ARDY pins HIGH. Whenever a refresh cycle occurs outside of a predefined internal wait state area, the external ready pins, which are active HIGH, will complement the bus cycle.

If it is necessary to use the external ready signals for certain memory regions, then it will be necessary to add logic which will generate a ready signal whenever the address of a refresh cycle falls where there is no memory. This can easily be accomplished by either decoding a couple of high order address lines or

In many low-cost 80186/80188 designs, dynamic memory offers an excellent cost/performance advantage. However, DRAM interfacing is often complicated by the need to perform memory refreshing. This application brief describes how to use the Timer and DMA functionality of the 80186/80188 to perform memory refresh.

THEORY OF OPERATION

Dynamic RAM refreshing is accomplished by strobing a ROW address to every ROW of the DRAM within a given period of time. One way to do this is to perform periodic sequential reads to the DRAM using a DMA controller and a Timer. This can be achieved with the 80186/188 by Programming Timer 2 and one of the DMA channels such that the timer generated one DMA cycle approximately every 15 micro-seconds. Please note that this is a single row refresh method and not a burst refresh. Single row refreshing reduces the bus overhead considerably when compared to burst refreshing.

The control logic of the DRAM is such that a RAS (row address strobe) occurs on every memory read, regardless of the address. This is necessary because the DMA channel is cycling through the entire 1 MByte address space and the address of the refresh cycle does not always fall within the range of the DRAM bank.

Although the address may be outside the DRAM range, the lower address bits continue to change and roll over to provide the row address.

READY LOGIC WITH MEMORY

Since the DMA controller is cycling through the entire 1 MByte address space, care must be taken to ensure that a READY signal is available for all addresses. One way to do this is to use only the internal wait state generator for memory areas and to strap the SRDY and ARDY pins HIGH. Whenever a refresh cycle occurs outside of a predefined internal wait state area, the external ready pins, which are active HIGH, will complete the bus cycle.

If it is necessary to use the external ready signals for certain memory regions, then it will be necessary to add logic which will generate a ready signal whenever the address of a refresh cycle falls where there is no memory. This can easily be accomplished by either decoding a couple of high order address lines, or by AND-ing

all the chip selects so that READY goes active whenever all the memory chip selects are inactive (i.e. the cycle is not in a valid memory region).

BUS OVERHEAD

The absolute maximum overhead can be calculated at a given speed by taking the number of refresh cycles divided by the total number of bus cycles for a given period of time. At 8 MHz these values can be calculated as follows:

$$\frac{2 \text{ bus cycles}}{15.2 \mu\text{s}/500 \text{ ns}} \times 100 = 6.6\% \text{ maximum overhead}$$

In reality, the bus overhead associated with the DMA cycles is much lower due to the instruction prefetch queue. When a DMA cycle is requested by the timer for a refresh cycle, the Bus Interface Unit honors the request on the next bus cycle boundary (with the exception of LOCKed bus cycles and odd aligned accesses). Typically this time is idle time on the bus and the impact on the overall performance is extremely small. The following table shows more realistic data which was acquired by running 6 different benchmarks with and without the DMA channel enabled to provide refresh every 15.2 μs .

BENCHMARK RESULTS @ 8 MHz

	Minimum	Maximum	Average
80186	1.3%	5.9%	2.5%
80188	2.4%	6.5%	3.4%

The programs which showed the highest bus overhead tended to be very bus intensive. Also note that at faster frequencies the bus overhead becomes even less.

DMA OPERATION

The DMA controller is programmed to be source synchronized with the TC (transfer count) bit cleared. This ensures that the DMA controller never reaches a final count. The source pointer continues to increment through memory on every cycle. When FFFFH is reached, the address rolls over to 0000H.

The programming values for the DMA registers are shown in Figure 1. The source pointer may be initialized to any location since the starting location of the refresh is arbitrary.

The value of the Transfer Count register is also arbitrary since the TC bit is not set. The DMA channel will continue to run cycles upon request from Timer 2 even after the Transfer Count register has reached zero. Once zero is reached, the Transfer Count register will roll over to FFFFH and continue to count down.

The destination pointer may be set to any available memory or I/O location. This pointer must be set so that it neither increments nor decrements. Otherwise, the address of the deposit cycle would cycle through memory or I/O doing writes which could possibly be destructive. Thus the INC and DEC bits of the control register should be cleared.

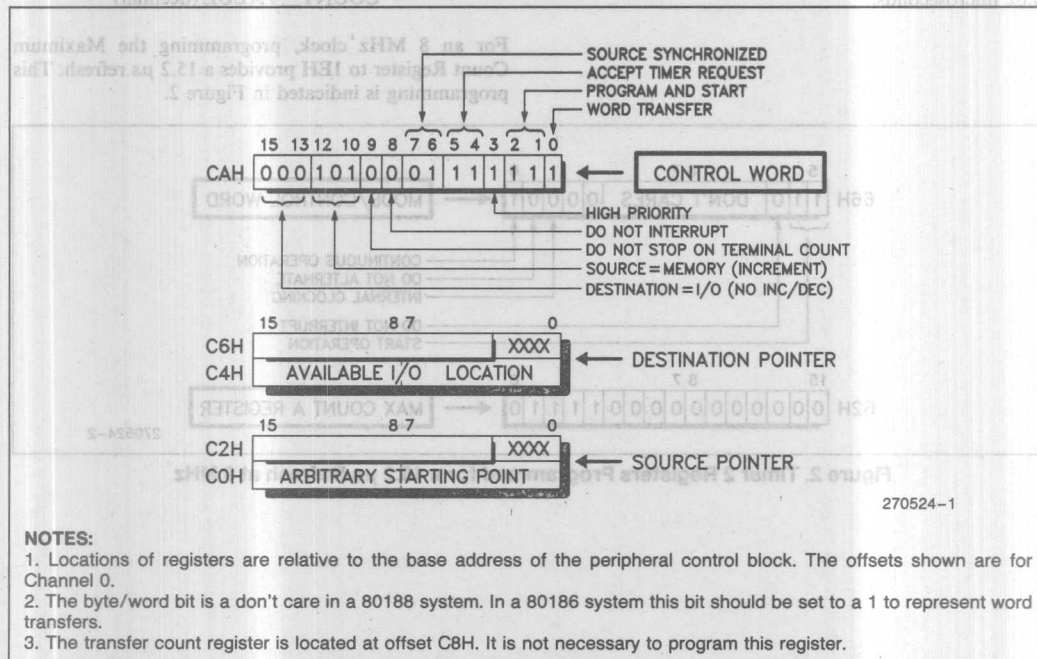


Figure 1. DMA Registers

TIMER OPERATION

Timer 2 must be programmed to generate a DMA request every time a row must be refreshed. Since we are not using a burst refresh, the refresh time is divided up evenly among the number of rows. For a 2 ms refresh DRAM with 128 rows, the time between rows equals 15.62 microseconds.

When setting the count value of the timer, keep in mind the timer clock is operating at one-fourth the CPU clock frequency. Thus, the equation for setting the timer count is:

$$\frac{(\text{CPU CLOUT FREQ}) \times (\text{Time Between ROWS})}{4} = \text{COUNT_VALUE (decimal)}$$

For an 8 MHz clock, programming the Maximum Count Register to 1EH provides a 15.2 μ s refresh. This programming is indicated in Figure 2.

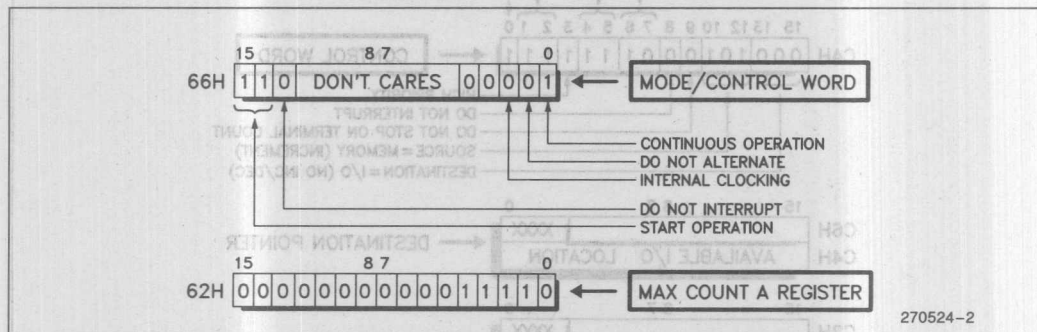


Figure 2. Timer 2 Registers Programmed for a 15.2 μ s Refresh at 8 MHz

EXAMPLE 1: DRAM CONTROL WITH A DELAY LINE

This is the most straight forward way of implementing the RAS and CAS logic. A RAS signal is generated by either $\overline{\text{RD}}$ or $\overline{\text{WR}}$ going active while the address is within the corresponding range. Normally the logic for $\overline{\text{RAS}}$ would also go active for a refresh cycle status, but since this information is not available on the 80186/80188, a RAS must be generated for every $\overline{\text{RD}}$ and $\overline{\text{WR}}$, regardless address.

The $\overline{\text{MUX}}$ signal is used to change from the $\overline{\text{RAS}}$ address to the $\overline{\text{CAS}}$ address after latching with RAS . This is accomplished by using a delay line which generates a $\overline{\text{MUX}}$ signal by a fixed number of nano-seconds after $\overline{\text{RAS}}$ is generated. The important timing here is the necessary hold time for the row address into the DRAM .

The $\overline{\text{MUX}}$ signal is initially HIGH which sends the A side (see Figure 3) Row address through the multiplex-

er to the DRAM. This address consists of A0 through A7. The B address (A8 through A16) is selected when $\overline{\text{MUX}}$ goes LOW. The system shown in Figure 3 represents that of an 80188 system.

For an 80186 system, the A address would start at A1. The least significant address line A0 along with BHE would be used to decode WE into WEH and WEL which will be shown in the second example. Also, the 186 DMA must be set to do word transfers so that the address is incremented by 2 after each refresh cycle. This is necessary to ensure A1 increments by 1 every refresh cycle.

CAS is generated in the same manner by delaying the MUX signal a fixed number of nano-seconds. Typically CAS goes inactive at the same time as RAS to ensure a valid CAS precharge time before the next DRAM access. The 80186/188 chip selects are used to ensure that CAS only goes active when the address falls within the DRAM bank range, and to ensure that CAS does not go active during I/O cycles.

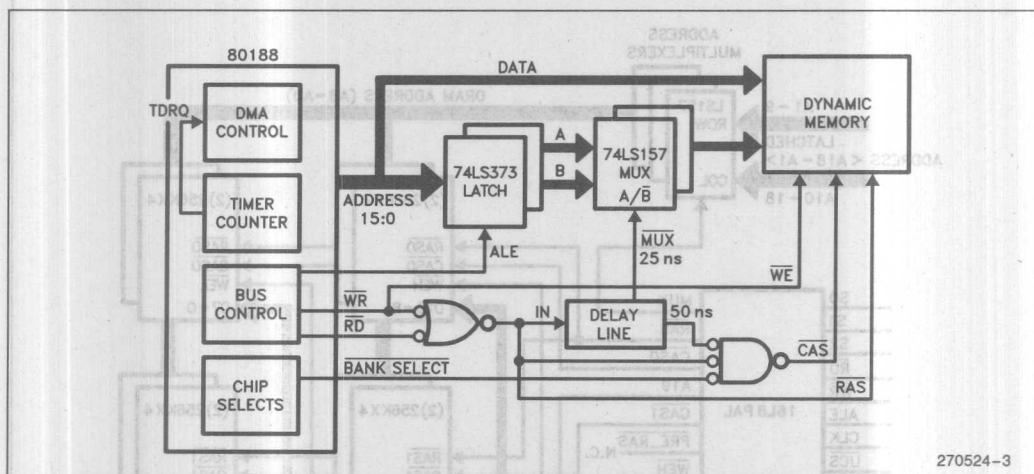


Figure 3. Using A Delay Line for DRAM Control

EXAMPLE 2: DRAM CONTROL WITH A PAL*

This design uses a PAL to generate all the control logic for the DRAM array. Internal feedback is used on the signals to control the timing and states of the $\overline{\text{RAS}}$, $\overline{\text{MUX}}$ and $\overline{\text{CAS}}$ signals.

This design uses 256k X 4 DRAMs. With minor changes to the PAL equations this design could just as easily make use of 64k X 1, 64k X 4, or 256k X 1 DRAMs.

The $\overline{\text{RAS}}$ signal is generated off ALE going LOW, bus cycle status active, and $\overline{\text{PRE_RAS}}$ being active. The $\overline{\text{PRE_RAS}}$ signal is necessary to ensure that a $\overline{\text{RAS}}$ is not accidentally generated when S2-S0 are becoming valid and ALE has not yet gone HIGH in T4 phase 2. $\overline{\text{PRE_RAS}}$ does not go active until ALE has gone HIGH.

$\overline{\text{RAS}}$ is initiated for every memory read and write regardless of the bus cycle address. This ensures a row

refresh when the refresh address falls outside of the DRAM bank and also a refresh to both banks simultaneously so that the frequency of the refresh can be set for the number of rows in one bank of DRAM.

The $\overline{\text{UCS}}$ (Upper Chip Select) from the 80186/188 is used to disable DRAM signals when the processor is attempting to access upper memory control ROM. Thus the portion of memory used by the $\overline{\text{UCS}}$ (maximum 256k) is unavailable in the upper DRAM. However, the $\overline{\text{RAS}}$ signal must still be allowed during $\overline{\text{UCS}}$ access to ensure refreshing when the DMA refresh cycle occurs in the $\overline{\text{UCS}}$ region.

$\overline{\text{MUX}}$ is generated off T2 phase 1 and $\overline{\text{RAS}}$ active. $\overline{\text{MUX}}$ will remain low until the current $\overline{\text{RAS}}$ signal goes inactive during T3 phase 2.

$\overline{\text{CAS0}}$ and $\overline{\text{CAS1}}$ are generated off $\overline{\text{MUX}}$ being active and T2 phase 2 of the bus cycle. $\overline{\text{CAS}}$ goes inactive at the start of T4 phase 2.

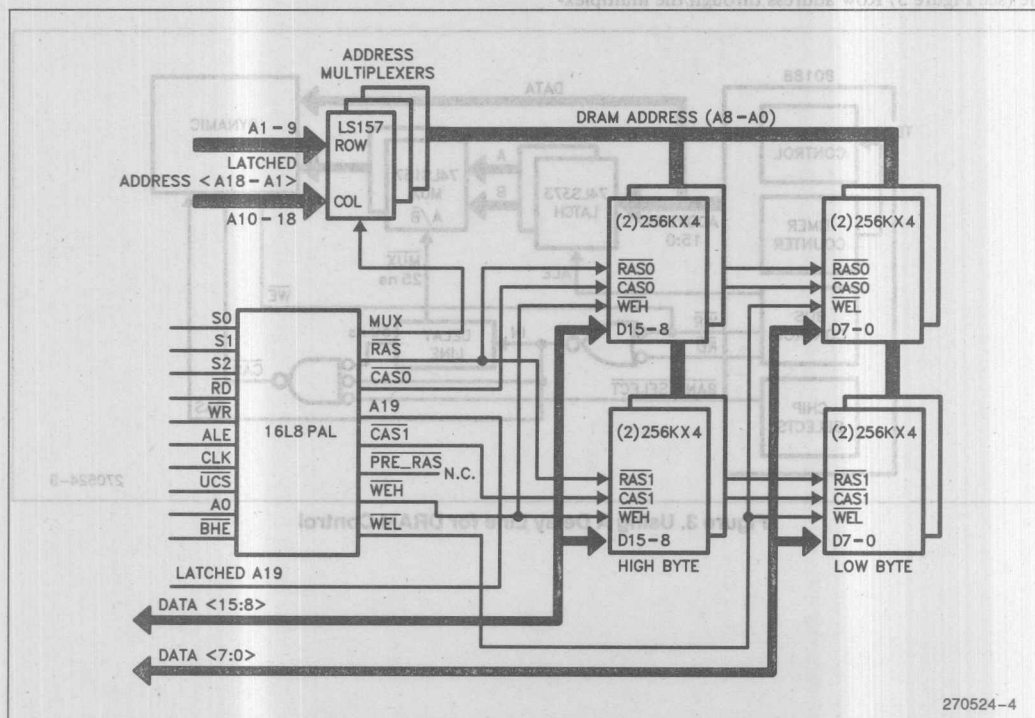


Figure 4. Using a PAL for DRAM Control

*PAL® is a registered trademark of Monolithic Memories.

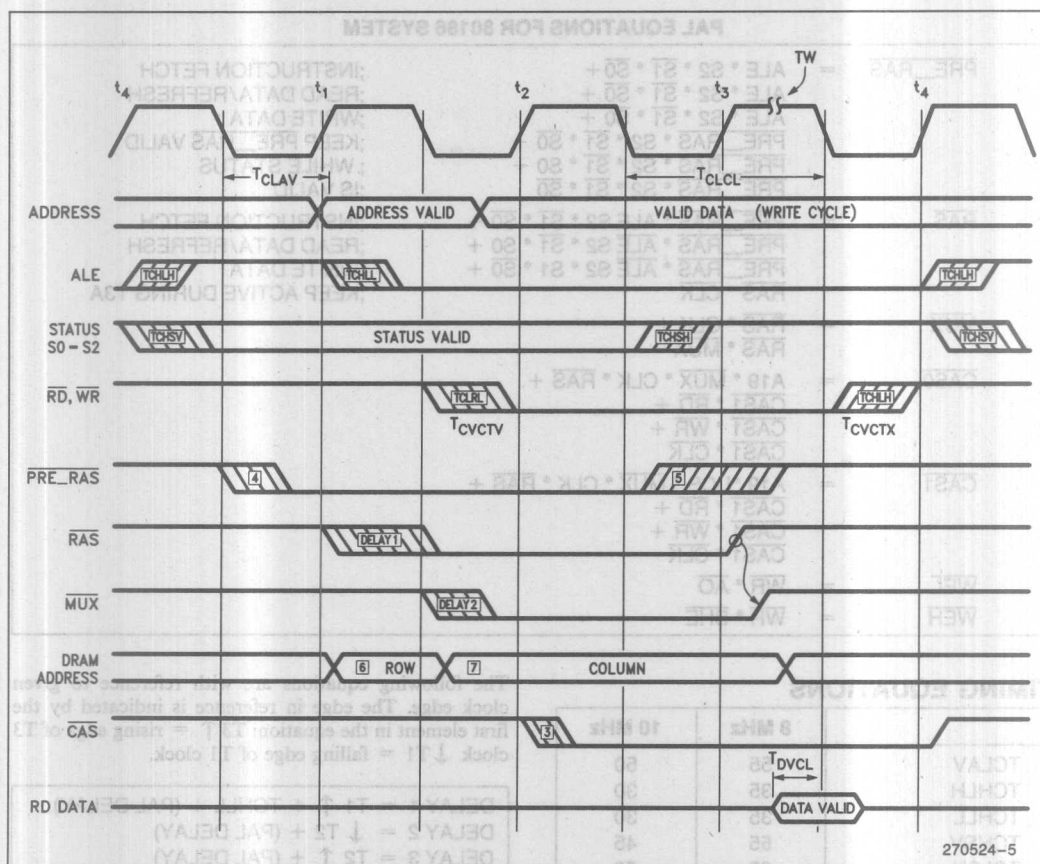


Figure 5. Timing Diagram for PAL DRAM Controller

PAL EQUATIONS FOR 80186 SYSTEM

$\overline{\text{PRE_RAS}}$	=	$\text{ALE} * \text{S2} * \overline{\text{S1}} * \overline{\text{S0}} +$ $\text{ALE} * \text{S2} * \overline{\text{S1}} * \overline{\text{S0}} +$ $\text{ALE} * \text{S2} * \overline{\text{S1}} * \overline{\text{S0}} +$ $\text{PRE_RAS} * \text{S2} * \overline{\text{S1}} * \overline{\text{S0}} +$ $\text{PRE_RAS} * \text{S2} * \overline{\text{S1}} * \overline{\text{S0}} +$ $\text{PRE_RAS} * \text{S2} * \overline{\text{S1}} * \overline{\text{S0}}$;INSTRUCTION FETCH ;READ DATA/REFRESH ;WRITE DATA ;KEEP PRE_RAS VALID ;WHILE STATUS ;IS VALID
RAS	=	$\text{PRE_RAS} * \text{ALE} * \text{S2} * \overline{\text{S1}} * \overline{\text{S0}} +$ $\text{PRE_RAS} * \text{ALE} * \text{S2} * \overline{\text{S1}} * \overline{\text{S0}} +$ $\text{PRE_RAS} * \text{ALE} * \text{S2} * \overline{\text{S1}} * \overline{\text{S0}} +$ $\text{RAS} * \text{CLK}$;INSTRUCTION FETCH ;READ DATA/REFRESH ;WRITE DATA ;KEEP ACTIVE DURING T3A
MUX	=	$\text{RAS} * \text{CLK} +$ $\text{RAS} * \text{MUX}$	
$\overline{\text{CAS0}}$	=	$\text{A19} * \text{MUX} * \text{CLK} * \overline{\text{RAS}} +$ $\text{CAS1} * \overline{\text{RD}} +$ $\text{CAS1} * \overline{\text{WR}} +$ $\text{CAS1} * \text{CLK}$	
$\overline{\text{CAS1}}$	=	$\text{A19} * \text{UCS} * \text{MUX} * \text{CLK} * \overline{\text{RAS}} +$ $\text{CAS1} * \overline{\text{RD}} +$ $\text{CAS1} * \overline{\text{WR}} +$ $\text{CAS1} * \text{CLK}$	
$\overline{\text{WEL}}$	=	$\overline{\text{WR}} * \overline{\text{A0}}$	
$\overline{\text{WEH}}$	=	$\overline{\text{WR}} * \overline{\text{BHE}}$	

TIMING EQUATIONS

	8 MHz	10 MHz
TCLAV	55	50
TCHLH	35	30
TCHLL	35	30
TCHSV	55	45
TCLSH	65	50
TCLRL/TCVCTV	70	56
TCLRH	55	44
TDVCL	20	15

The following equations are with reference to given clock edge. The edge in reference is indicated by the first element in the equation: T3 ↑ = rising edge of T3 clock ↓ T1 = falling edge of T1 clock.

DELAY 1	=	T1 ↑ + TCHLL + (PAL DELAY)
DELAY 2	=	↓ T2 + (PAL DELAY)
DELAY 3	=	T2 ↑ + (PAL DELAY)
DELAY 4	=	↓ T1 + (PAL DELAY)
DELAY 5	=	↓ T3 + TCLSH + (PAL DELAY)
DELAY 6	=	↓ T1 + TCLAV + (MUX DELAY)
DELAY 7	=	↓ T2 + DELAY 2 + (MUX DELAY)

ACCESS TIME FROM $\overline{\text{RAS}}$ = 2.5 (TCLCL) - DELAY 1 - TDVCL
ACCESS TIME FROM $\overline{\text{CAS}}$ = 1.5 (TCLCL) - DELAY 3 - TDVCL

**MCS-70 Application Notes &
Article Reprint**

b

2.1 General Description

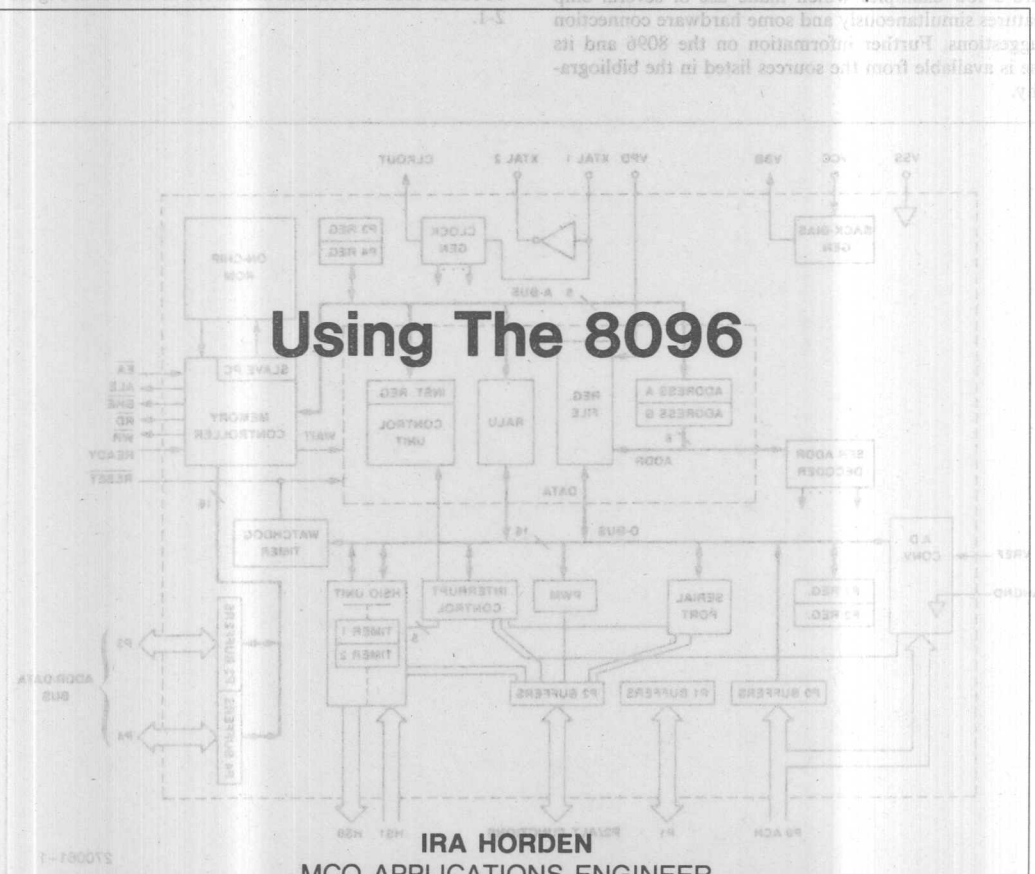
Unlike microprocessors, microcontrollers are generally optimized for specific applications. Intel's 8096 was optimized for general control tasks while the 8051 was optimized for 8-bit math and single bit boolean operations. The 8096 has been designed for high speed/high performance control applications. Because it has been designed for these applications the 8096 architecture is different from that of the 8048 or 8051.

There are two major sections of the 8096: the CPU and the I/O section. Each of these sections can be subdivided into functional blocks as shown in Figure 2-1.

High speed digital signals are frequently encountered in modern control applications. In addition, there is often a requirement for high speed 16-bit and 32-bit precision calculations. The MCS-96 product line, generally referred to as the 8096, is designed to be used in applications which require high speed calculations and fast I/O operations.

The 8096 is a 16-bit microcontroller with dedicated I/O subsystems and a complete set of 16-bit arithmetic instructions including multiply and divide operations. This App-note will briefly describe the 8096 in section 2, and then give short examples of how to use each of its major features in section 3. The concluding sections (section 4) which make use of several chip features (interrupts, and some hardware connection techniques) are available from the sources listed in the bibliography.

September 1987



1.0 INTRODUCTION

High speed digital signals are frequently encountered in modern control applications. In addition, there is often a requirement for high speed 16-bit and 32-bit precision in calculations. The MCS[®]-96 product line, generically referred to as the 8096, is designed to be used in applications which require high speed calculations and fast I/O operations.

The 8096 is a 16-bit microcontroller with dedicated I/O subsystems and a complete set of 16-bit arithmetic instructions including multiply and divide operations. This Ap-note will briefly describe the 8096 in section 2, and then give short examples of how to use each of its key features in section 3. The concluding sections feature a few examples which make use of several chip features simultaneously and some hardware connection suggestions. Further information on the 8096 and its use is available from the sources listed in the bibliography.

2.0 8096 OVERVIEW

2.1. General Description

Unlike microprocessors, microcontrollers are generally optimized for specific applications. Intel's 8048 was optimized for general control tasks while the 8051 was optimized for 8-bit math and single bit boolean operations. The 8096 has been designed for high speed/high performance control applications. Because it has been designed for these applications the 8096 architecture is different from that of the 8048 or 8051.

There are two major sections of the 8096; the CPU section and the I/O section. Each of these sections can be subdivided into functional blocks as shown in Figure 2-1.

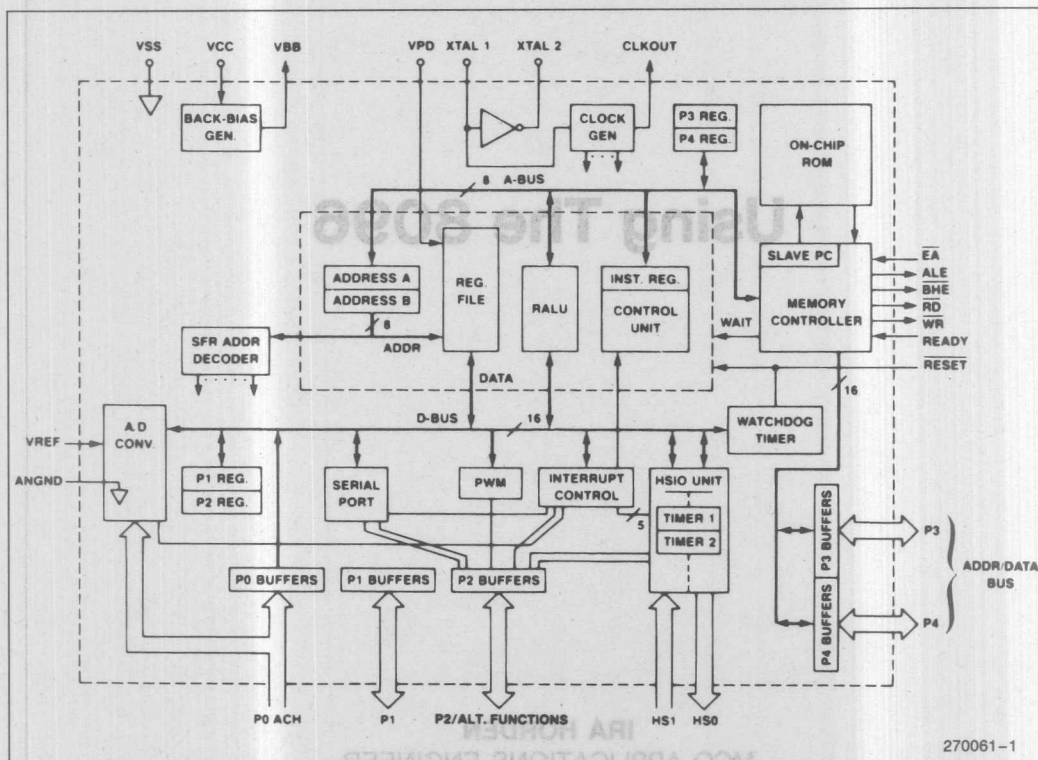


Figure 2-1. 8096 Block Diagram

2.1.1.1. CPU SECTION

The CPU of the 8096 uses a 16-bit ALU which operates on a 256-byte register file instead of an accumulator. Any of the locations in the register file can be used for sources or destinations for most of the instructions. This is called a register to register architecture. Many of the instructions can also use bytes or words from anywhere in the 64K byte address space as operands. A memory map is shown in Figure 2-2.

In the lower 24 bytes of the register file are the register-mapped I/O control locations, also called Special Function Registers or SFRs. These registers are used to control the on-chip I/O features. The remaining 232 bytes are general purpose RAM, the upper 16 of which can be kept alive using a low current power-down mode.

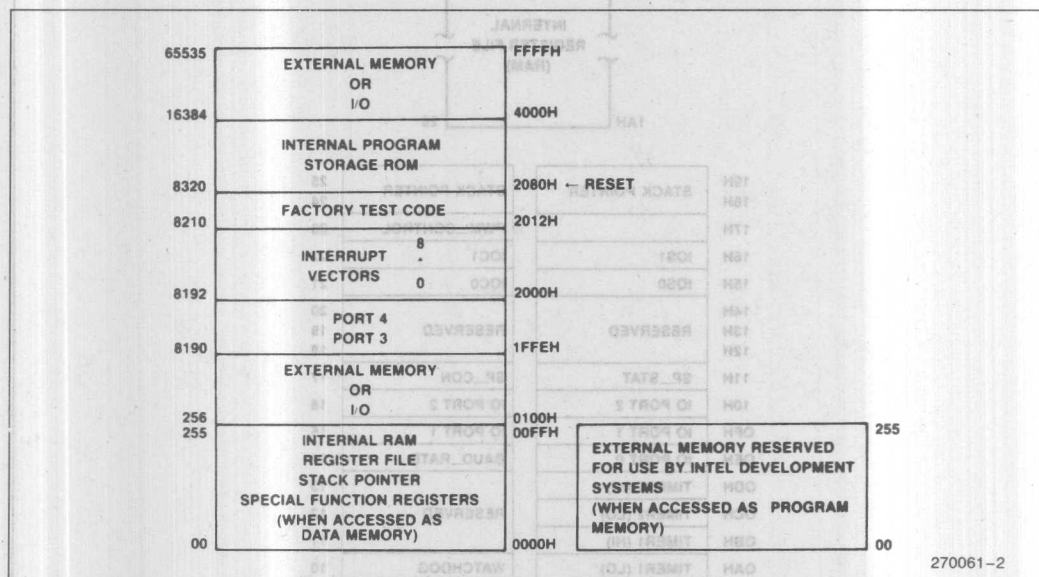


Figure 2-2. Memory Map

Figure 2-3 shows the layout of the register mapped I/O. Some of these registers serve two functions, one if they are read from and another if they are written

to. More information about the use of these registers is included in the description of the features which they control.

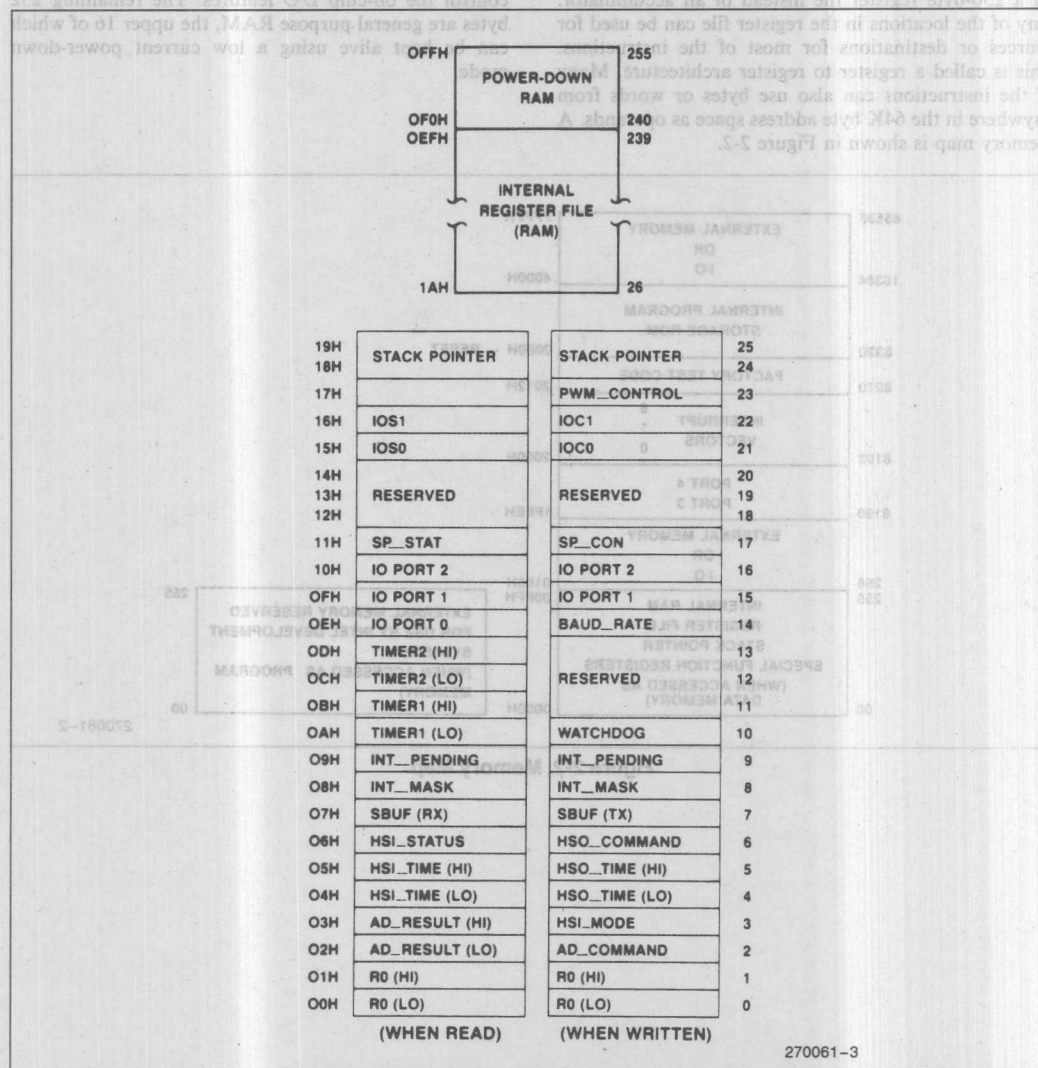


Figure 2-3: SFR Layout

2.1.2. I/O FEATURES

Many of the I/O features on the 8096 are designed to operate with little CPU intervention. A list of the major I/O functions is shown in Figure 2-4. The Watchdog Timer is an internal timer which can be used to reset the system if the software fails to operate properly. The Pulse-Width-Modulation (PWM) output can be used as a rough D to A, a motor driver, or for many other purposes. The A to D converter (ADC) has 8 multiplexed inputs and 10-bit resolution. The serial port has several modes and its own baud rate generator. The High Speed I/O section includes a 16-bit timer, a 16-bit counter, a 4-input programmable edge detector, 4 software timers, and a 6-output programmable event generator. All of these features will be described in section 2.3.

2.2. The Processor Section

2.2.1. OPERATIONS AND ADDRESSING MODES

The 8096 has 100 instructions, some of which operate on bits, some on bytes, some on words and some on longs (double words). All of the standard logical and arithmetic functions are available for both byte and word operations. Bit operations and long operations are provided for some instructions. There are also flag manipulation instructions as well as jump and call instructions. A full set of conditional jumps has been included to speed up testing for various conditions.

Bit operations are provided by the Jump Bit and Jump Not Bit instructions, as well as by immediate masking of bytes. These bit operations can be performed on any of the bytes in the register file or on any of the special function registers. The fast bit manipulation of the SFRs can provide rapid I/O operations.

A symmetric set of byte and word operations make up the majority of the 8096 instruction set. The assembly language for the 8096 (ASM-96) uses a "B" suffix on a mnemonic to indicate a byte operation, without this suffix a word operation is indicated. Many of these operations can have one, two or three operands. An example of a one operand instruction would be:

NOT Value1 ; Value1 := 1's complement (Value1)

A two operand instruction would have the form:

ADD Value2,Value1 ; Value2 := Value2 + Value1

A three operand instruction might look like:

MUL Value3,Value2,Value1 ;
Value3 := Value2 * Value1

The three operand instructions combined with the register to register architecture almost eliminate the necessity of using temporary registers. This results in a faster processing time than machines that have equivalent instruction execution times, but use a standard architecture.

Long (32-bit) operations include shifts, normalize, and multiply and divide. The word divide is a 32-bit by 16-bit operation with a 16-bit quotient and 16-bit remainder. The word multiply is a word by word multiply with a long result. Both of these operations can be done in either the signed or unsigned mode. The direct unsigned modes of these instructions take only 6.5 microseconds. A normalize instruction and sticky bit flag have been included in the instruction set to provide hardware support for the software floating point package (FPAL-96).

Major I/O Functions	
High Speed Input Unit	Provides Automatic Recording of Events
High Speed Output Unit	Provides Automatic Triggering of Events and Real-Time Interrupts
Pulse Width Modulation	Output to Drive Motors or Analog Circuits
A to D Converter	Provides Analog Input
Watchdog Timer	Resets 8096 if a Malfunction Occurs
Serial Port	Provides Synchronous or Asynchronous Link
Standard I/O Lines	Provide Interface to the External World when other Special Features are not needed

Figure 2-4. Major I/O Functions

Mnemonic	Operands	Operation (Note 1)	Flags						Notes
			Z	N	C	V	VT	ST	
ADD/ADDB	2	$D \leftarrow D + A$	✓	✓	✓	✓	↑	—	
ADD/ADDB	3	$D \leftarrow B + A$	✓	✓	✓	✓	↑	—	
ADDC/ADDCB	2	$D \leftarrow D + A + C$	↓	✓	✓	✓	↑	—	
SUB/SUBB	2	$D \leftarrow D - A$	✓	✓	✓	✓	↑	—	
SUB/SUBB	3	$D \leftarrow B - A$	✓	✓	✓	✓	↑	—	
SUBC/SUBCB	2	$D \leftarrow D - A + C - 1$	↓	✓	✓	✓	↑	—	
CMP/CMPB	2	$D - A$	✓	✓	✓	✓	↑	—	
MUL/MULU	2	$D, D + 2 \leftarrow D * A$	—	—	—	—	—	?	2
MUL/MULU	3	$D, D + 2 \leftarrow B * A$	—	—	—	—	—	?	2
MULB/MULUB	2	$D, D + 1 \leftarrow D * A$	—	—	—	—	—	?	3
MULB/MULUB	3	$D, D + 1 \leftarrow B * A$	—	—	—	—	—	?	3
DIVU	2	$D \leftarrow (D, D + 2)/A, D + 2 \leftarrow \text{remainder}$	—	—	—	✓	↑	—	2
DIVUB	2	$D \leftarrow (D, D + 1)/A, D + 1 \leftarrow \text{remainder}$	—	—	—	✓	↑	—	3
DIV	2	$D \leftarrow (D, D + 2)/A, D + 2 \leftarrow \text{remainder}$	—	—	—	?	↑	—	2
DIVB	2	$D \leftarrow (D, D + 1)/A, D + 1 \leftarrow \text{remainder}$	—	—	—	?	↑	—	3
AND/ANDB	2	$D \leftarrow D \text{ and } A$	✓	✓	0	0	—	—	
AND/ANDB	3	$D \leftarrow B \text{ and } A$	✓	✓	0	0	—	—	
OR/ORB	2	$D \leftarrow D \text{ or } A$	✓	✓	0	0	—	—	
XOR/XORB	2	$D \leftarrow D \text{ (excl. or) } A$	✓	✓	0	0	—	—	
LD/LDB	2	$D \leftarrow A$	—	—	—	—	—	—	
ST/STB	2	$A \leftarrow D$	—	—	—	—	—	—	
LDBSE	2	$D \leftarrow A; D + 1 \leftarrow \text{SIGN}(A)$	—	—	—	—	—	—	3, 4
LDBZE	2	$D \leftarrow A; D + 1 \leftarrow 0$	—	—	—	—	—	—	3, 4
PUSH	1	$SP \leftarrow SP - 2; (SP) \leftarrow A$	—	—	—	—	—	—	
POP	1	$A \leftarrow (SP); SP \leftarrow SP + 2$	—	—	—	—	—	—	
PUSHF	0	$SP \leftarrow SP - 2; (SP) \leftarrow \text{PSW};$ $\text{PSW} \leftarrow 0000\text{H}$	0	0	0	0	0	0	
POPF	0	$\text{PSW} \leftarrow (SP); SP \leftarrow SP + 2; I \leftarrow 0$	✓	✓	✓	✓	✓	✓	
SJMP	1	$PC \leftarrow PC + 11\text{-bit offset}$	—	—	—	—	—	—	5
LJMP	1	$PC \leftarrow PC + 16\text{-bit offset}$	—	—	—	—	—	—	5
BR (indirect)	1	$PC \leftarrow (A)$	—	—	—	—	—	—	
SCALL	1	$SP \leftarrow SP - 2; (SP) \leftarrow PC;$ $PC \leftarrow PC + 11\text{-bit offset}$	—	—	—	—	—	—	5
LCALL	1	$SP \leftarrow SP - 2; (SP) \leftarrow PC;$ $PC \leftarrow PC + 16\text{-bit offset}$	—	—	—	—	—	—	5
RET	0	$PC \leftarrow (SP); SP \leftarrow SP + 2$	—	—	—	—	—	—	
J (conditional)	1	$PC \leftarrow PC + 8\text{-bit offset (if taken)}$	—	—	—	—	—	—	5
JC	1	Jump if C = 1	—	—	—	—	—	—	5
JNC	1	Jump if C = 0	—	—	—	—	—	—	5
JE	1	Jump if Z = 1	—	—	—	—	—	—	5

Figure 2-5. Instruction Summary

NOTES:

1. If the mnemonic ends in "B", a byte operation is performed, otherwise a word operation is done. Operands D, B, and A must conform to the alignment rules for the required operand type. D and B are locations in the register file; A can be located anywhere in memory.
2. D, D + 2 are consecutive WORDS in memory; D is DOUBLE-WORD aligned.
3. D, D + 1 are consecutive BYTES in memory; D is WORD aligned.
4. Changes a byte to a word.
5. Offset is a 2's complement number.

Mnemonic	Operands	Operation (Note 1)	Flags						Notes
			Z	N	C	V	VT	ST	
JNE	1	Jump if Z = 0	—	—	—	—	—	—	5
JGE	1	Jump if N = 0	—	—	—	—	—	—	5
JLT	1	Jump if N = 1	—	—	—	—	—	—	5
JGT	1	Jump if N = 0 and Z = 0	—	—	—	—	—	—	5
JLE	1	Jump if N = 1 or Z = 1	—	—	—	—	—	—	5
JH	1	Jump if C = 1 and Z = 0	—	—	—	—	—	—	5
JNH	1	Jump if C = 0 or Z = 1	—	—	—	—	—	—	5
JV	1	Jump if V = 1	—	—	—	—	—	—	5
JNV	1	Jump if V = 0	—	—	—	—	—	—	5
JVT	1	Jump if VT = 1; Clear VT	—	—	—	—	0	—	5
JNVT	1	Jump if VT = 0; Clear VT	—	—	—	—	0	—	5
JST	1	Jump if ST = 1	—	—	—	—	—	—	5
JNST	1	Jump if ST = 0	—	—	—	—	—	—	5
JBS	3	Jump if Specified Bit = 1	—	—	—	—	—	—	5, 6
JBC	3	Jump if Specified Bit = 0	—	—	—	—	—	—	5, 6
DJNZ	1	D ← D - 1; if D ≠ 0 then PC ← PC + 8-bit offset	—	—	—	—	—	—	5
DEC/DECB	1	D ← D - 1	✓	✓	✓	✓	↑	—	
NEG/NEGB	1	D ← 0 - D	✓	✓	✓	✓	↑	—	
INC/INCB	1	D ← D + 1	✓	✓	✓	✓	↑	—	
EXT	1	D ← D; D + 2 ← Sign (D)	✓	✓	0	0	—	—	2
EXTB	1	D ← D; D + 1 ← Sign (D)	✓	✓	0	0	—	—	3
NOT/NOTB	1	D ← Logical Not (D)	✓	✓	0	0	—	—	
CLR/CLRB	1	D ← 0	1	0	0	0	—	—	
SHL/SHLB/SHLL	2	C ← msb ———— lsb ← 0	✓	?	✓	✓	↑	—	7
SHR/SHRB/SHRL	2	0 → msb ———— lsb → C	✓	?	✓	0	—	✓	7
SHRA/SHRAB/SHRAL	2	msb → msb ———— lsb → C	✓	✓	✓	0	—	✓	7
SETC	0	C ← 1	—	—	1	—	—	—	
CLRC	0	C ← 0	—	—	0	—	—	—	
CLRVT	0	VT ← 0	—	—	—	—	0	—	
RST	0	PC ← 2080H	0	0	0	0	0	0	8
DI	0	Disable All Interrupts (I ← 0)	—	—	—	—	—	—	
EI	0	Enable All Interrupts (I ← 1)	—	—	—	—	—	—	
NOP	0	PC ← PC + 1	—	—	—	—	—	—	
SKIP	0	PC ← PC + 2	—	—	—	—	—	—	
NORML	2	Left Shift Till msb = 1; D ← shift count	✓	?	0	—	—	—	7
TRAP	0	SP ← SP - 2; (SP) ← PC PC ← (2010H)	—	—	—	—	—	—	9

Figure 2-5. Instruction Summary (Continued)

NOTES:

1. If the mnemonic ends in "B", a byte operation is performed, otherwise a word operation is done. Operands D, B, and A must conform to the alignment rules for the required operand type. D and B are locations in the register file; A can be located anywhere in memory.
5. Offset is a 2's complement number.
6. Specified bit is one of the 2048 bits in the register file.
7. The "L" (Long) suffix indicates double-word operation.
8. Initiates a Reset by pulling RESET low. Software should re-initialize all the necessary registers with code starting at 2080H.
9. The assembler will not accept this mnemonic.

One operand of most of the instructions can be used with any one of six addressing modes. These modes increase the flexibility and overall execution speed of the 8096. The addressing modes are: register-direct, immediate, indirect, indirect with auto-increment, and long and short indexed.

The fastest instruction execution is gained by using either register direct or immediate addressing. Register-direct addressing is similar to normal direct addressing, except that only addresses in the register file or SFRs can be addressed. The indexed mode is used to directly address the remainder of the 64K address space. Immediate addressing operates as would be expected, using the data following the opcode as the operand.

Both of the indirect addressing modes use the value in a word register as the address of the operand. If the indirect auto-increment mode is used then the word register is incremented by one after a byte access or by two after a word access. This mode is particularly useful for accessing lookup tables.

Access to any of the locations in the 64K address space can be obtained by using the long indexed addressing

mode. In this mode a 16-bit 2's complement value is added to the contents of a word register to form the address of the operand. By using the zero register as the index, ASM96 (the assembler) can accept "direct" addressing to any location. The zero register is located at 0000H and always has a value of zero. A short indexed mode is also available to save some time and code. This mode uses an 8-bit 2's complement number as the offset instead of a 16-bit number.

2.2.2. ASSEMBLY LANGUAGE

The multiple addressing modes of the 8096 make it easy to program in assembly language and provide an excellent interface to high level languages. The instructions accepted by the assembler consist of mnemonics followed by either addresses or data. A list of the mnemonics and their functions are shown in Figure 2-5. The addresses or data are given in different formats depending on the addressing mode. These modes and formats are shown in Figure 2-6.

Additional information on 8096 assembly language is available in the MCS-96 Macro Assembler Users Guide, listed in the bibliography.

Mnem	Dest or Src1	: One operand direct
Mnem	Dest, Src1	: Two operand direct
Mnem	Dest, Src1, Src2	: Three operand direct
Mnem	#Src1	: One operand immediate
Mnem	Dest, #Src1	: Two operand immediate
Mnem	Dest, Src1, #Src2	: Three operand immediate
Mnem	[addr]	: One operand indirect
Mnem	[addr] +	: One operand indirect auto-increment
Mnem	Dest, [addr]	: Two operand indirect
Mnem	Dest, [addr] +	: Two operand indirect auto-increment
Mnem	Dest, Src1, [addr]	: Three operand indirect
Mnem	Dest, Src1, [addr] +	: Three operand indirect auto-increment
Mnem	Dest, offs [addr]	: Two operand indexed (short or long)
Mnem	Dest, Src1, offs [addr]	: Three operand indexed (short or long)
Where: "Mnem" is the instruction mnemonic		
"Dest" is the destination register		
"Src1", "Src2" are the source registers		
"addr" is a register containing a value to be used in computing the address of an operand		
"offs" is an offset used in computing the address of an operand		

270061-B3

Figure 2-6. Instruction Format

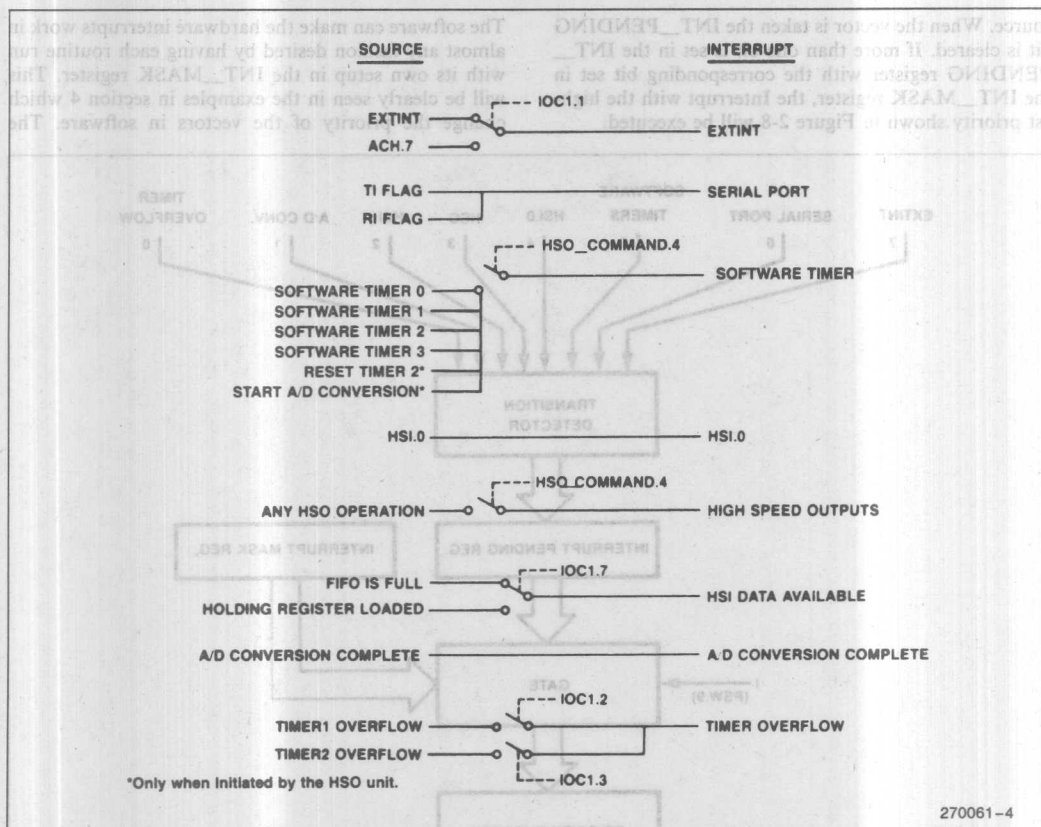


Figure 2-7. Interrupt Sources

2.2.3. INTERRUPTS

The flexibility of the instruction set is carried through into the interrupt system. There are 20 different interrupt sources that can be used on the 8096. The 20 sources vector through 8 locations or interrupt vectors. The vector names and their sources are shown in Figure 2-7, with their locations listed in Figure 2-8. Control of the interrupts is handled through the Interrupt Pending Register (INT_PENDING), the Interrupt Mask Register (INT_MASK), and the I bit in the PSW (PSW.9). Figure 2-9 shows a block diagram of the interrupt structure. The INT_PENDING register contains bits which get set by hardware when an interrupt occurs. If the interrupt mask register bit for that source is a 1 and PSW.9 = 1, a vector will be taken to the address listed in the interrupt vector table for that

Source	Vector Location		Priority
	(High Byte)	(Low Byte)	
Software	2011H	2010H	Not Applicable
Extint	200FH	200EH	7 (Highest)
Serial Port	200DH	200CH	6
Software Timers	200BH	200AH	5
HSI.0	2009H	2008H	4
High Speed Outputs	2007H	2006H	3
HSI Data Available	2005H	2004H	2
A/D Conversion Complete	2003H	2002H	1
Timer Overflow	2001H	2000H	0 (Lowest)

Figure 2-8. Interrupt Vectors and Priorities

bit is cleared. If more than one bit is set in the INT_PENDING register with the corresponding bit set in the INT_MASK register, the Interrupt with the highest priority shown in Figure 2-8 will be executed.

The software can make the hardware interrupts work in almost any fashion desired by having each routine run with its own setup in the INT_MASK register. This will be clearly seen in the examples in section 4 which change the priority of the vectors in software. The

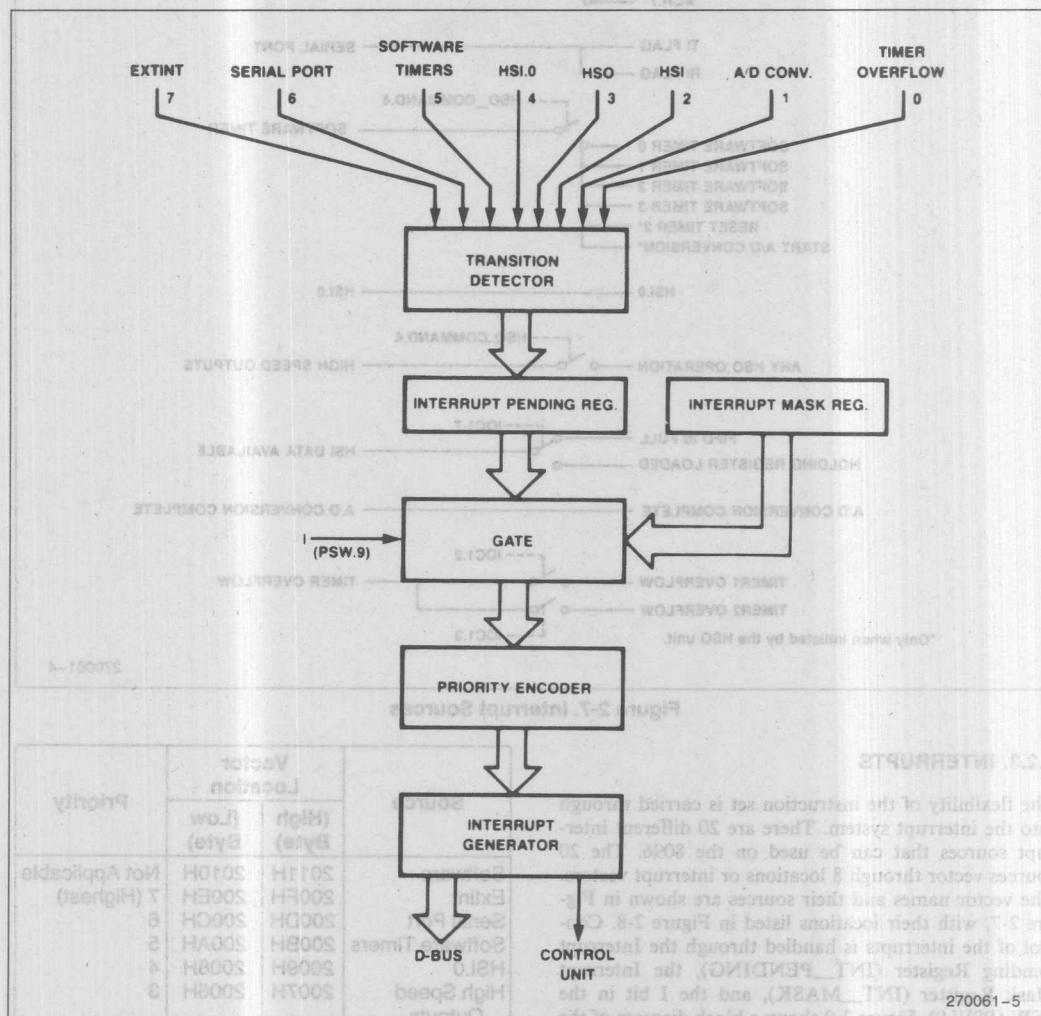


Figure 2-9. Interrupt Structure Block Diagram

15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
Z	N	V	VT	C	—	I	ST	INT_MASK							

WHERE:

Z is the zero flag. It is set when the result of an operation is zero.
N is the negative flag. It is set to the algebraically correct sign of the result regardless of overflows.
V is the overflow flag. It is set if an overflow occurs.
VT is the overflow trap flag. It is set when the VT flag is set and cleared by JVT, JNVT, or CLRVT.
C is the carry flag. It is set if a carry was generated by the prior operation.
I is the global interrupt enable bit.
ST is the sticky bit. It is set during a right shift if a one was shifted into and then out of the carry flag.
INT_MASK is the interrupt mask register and contains bits which individually enable the 8 interrupt vectors.

Figure 2-10. The PSW Register

PSW (shown in Figure 2-10), stores the INT_MASK register in its lower byte so that the mask register can be pushed and popped along with the machine status when moving in and out of routines. The action of pushing flags clears the PSW which includes PSW.9, the interrupt enable bit. Therefore, after a PUSHF instruction interrupts are disabled. In most cases an interrupt service routine will have the basic structure shown below.

INT VECTOR:

```

PUSHF
LDB INT_MASK, #xxxxxxxB
EI
- ;Insert service routine here
-
POPF
RET

```

The PUSHF instruction saves the PSW including the old INT_MASK register. The PSW, including the interrupt enable bit are left cleared. If some interrupts need to be enabled while the service routine runs, the INT_MASK is loaded with a new value and interrupts are globally enabled before the service routine continues. At the end of the service routine a POPF in-

struction is executed to restore the old PSW. The RET instruction is executed and the code returns to the desired location. Although the POPF instruction can enable the interrupts the next instruction will always execute. This prevents unnecessary building of the stack by ensuring that the RET always executes before another interrupt vector is taken.

2.3. On-Chip I/O Section

All of the on-chip I/O features of the 8096 can be accessed through the special function registers, as shown in Figure 2-3. The advantage of using register-mapped I/O is that these registers can be used as the sources or destinations of CPU operations. There are seven major I/O functions. Each one of these will be considered with a section of code to exemplify its usage. The first section covered will be the High Speed I/O, (HSIO), subsystem. This section includes the High Speed Input (HSI) unit, High Speed Output (HSO) unit, and the Timer/Counter section.

2.3.1. TIMER/COUNTERS

The 8096 has two time bases, Timer 1 and Timer 2. Timer 1 is a 16-bit free running timer which is incremented every 8 state times. (A state time is 3 oscillator periods, or 0.25 microseconds with a 12 MHz crystal.)

microsecond resolution. It can look for one of four events on each of four lines simultaneously, based on the information in the HSI Mode register, shown in Figure 2-12. The information is then stored in a seven level FIFO for later retrieval. Whenever the FIFO contains information, the earliest entry is placed in the holding register. When the holding register is read, the next valid piece of information is loaded into it. Interrupts can be generated by the HSI unit at the time the

holding register is loaded or when the FIFO has six or more entries.

2.3.3. HSO

Just as the HSI can be thought of as a message taker, the HSO can be thought of as a message sender. At times determined by the software, the HSO sends mes-

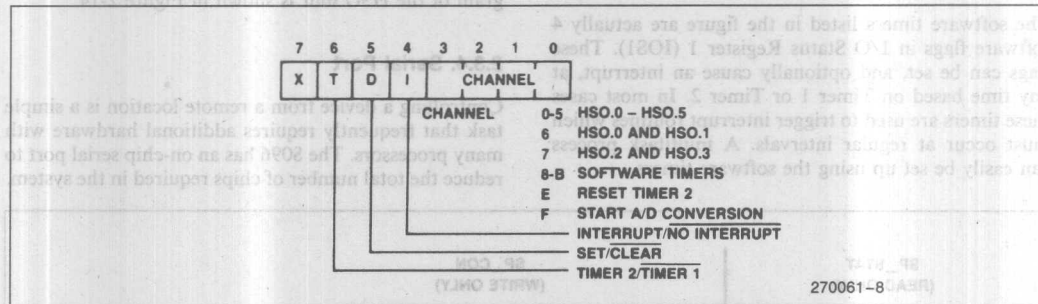


Figure 2-13. HSO Command Register

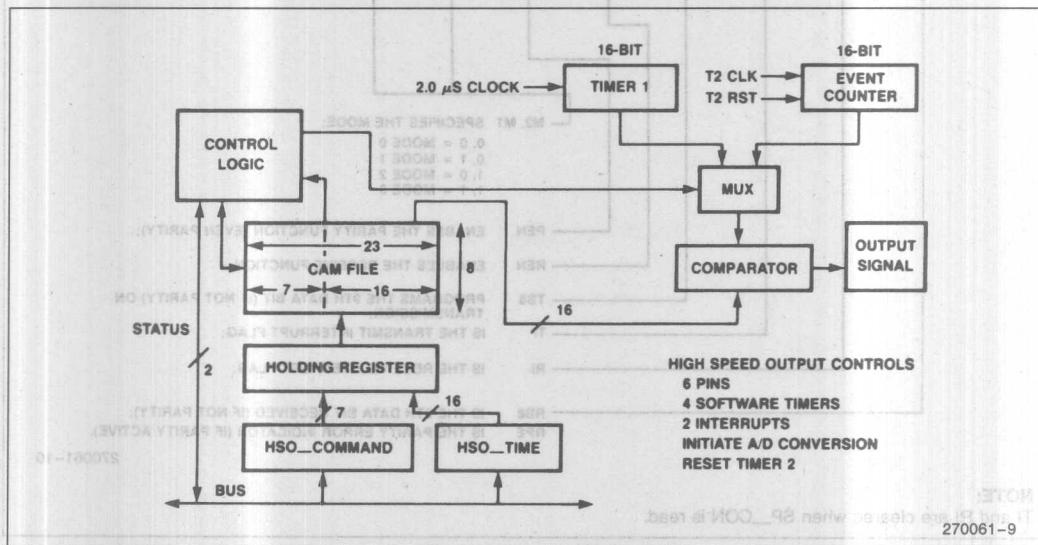


Figure 2-14. HSO Block Diagram

sages to various devices to have them turn on, turn off, start processing, or reset. Since the programmed times can be referenced to either Timer 1 or Timer 2, the HSO makes the two timers look like many. For example, if several events have to occur at specific times, the HSO unit can schedule all of the events based on a single timer. The events that can be scheduled to occur and the format of the command written to the HSO Command register are shown in Figure 2-13.

The software timers listed in the figure are actually 4 software flags in I/O Status Register 1 (IOS1). These flags can be set, and optionally cause an interrupt, at any time based on Timer 1 or Timer 2. In most cases these timers are used to trigger interrupt routines which must occur at regular intervals. A multitask process can easily be set up using the software timers.

A CAM (Content Addressable Memory) file is the main component of the HSO. This file stores up to eight events which are pending to occur. Every state time one location of the CAM is compared with the two timers. After 8 state times, (two microseconds with a 12 MHz clock), the entire CAM has been searched for time matches. If a match occurs the specified event will be triggered and that location of the CAM will be made available for another pending event. A block diagram of the HSO unit is shown in Figure 2-14.

2.3.4. Serial Port

Controlling a device from a remote location is a simple task that frequently requires additional hardware with many processors. The 8096 has an on-chip serial port to reduce the total number of chips required in the system.

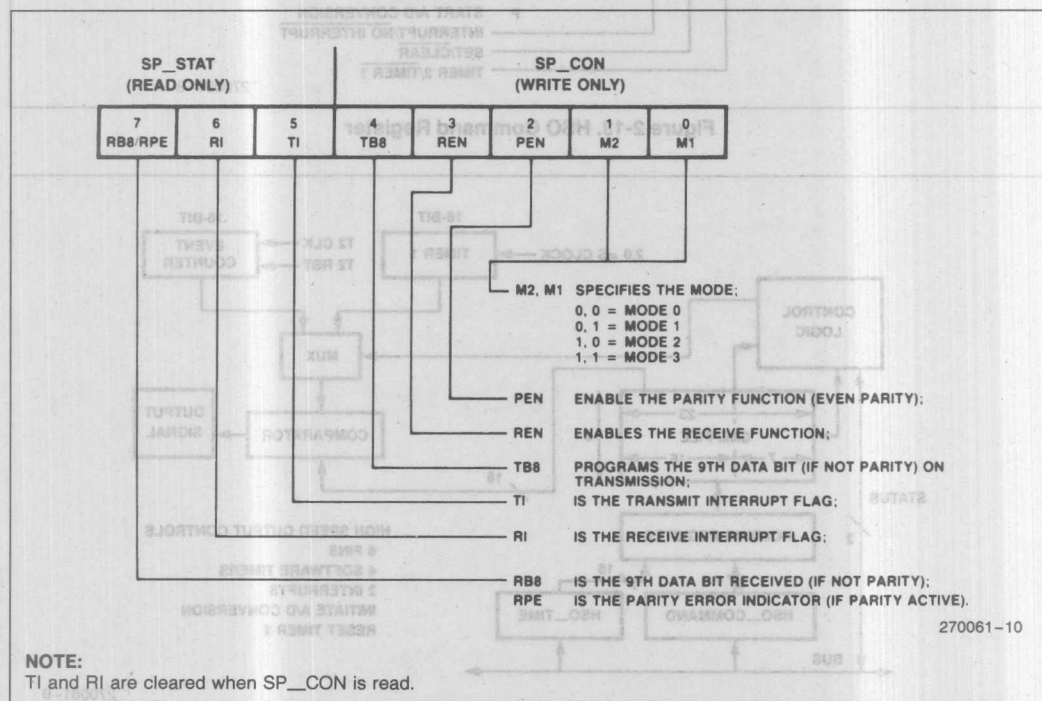


Figure 2-15. Serial Port Control/Status Register

The serial port is similar to that on the MCS-51 product line. It has one synchronous and three asynchronous modes. In the asynchronous modes baud rates of up to 187.5 Kbaud can be used, while in the synchronous mode rates up to 1.5 Mbaud are available. The chip has a baud rate generator which is independent of Timer 1 and Timer 2, so using the serial port does not take away any of the HSI, HSO or timer flexibility or functionality.

Control of the serial port is provided through the SPCON/SPSTAT (Serial Port CONTROL/Serial Port STATUS) register. This register, shown in Figure 2-15, has some bits which are read only and others which are write only. Although the functionality of the port is similar to that of the 8051, the names of some of the modes and control bits are different. The way in which the port is used from a software standpoint is also slightly different since RI and TI are cleared after each read of the register.

The four modes of the serial port are referred to as modes 0, 1, 2 and 3. Mode 0 is the synchronous mode, and is commonly used to interface to shift registers for I/O expansion. In this mode the port outputs a pulse train on the TXD pin and either transmits or receives data on the RXD pin. Mode 1 is the standard asynchronous mode, 8 bits plus a stop and start bit are sent or received. Modes 2 and 3 handle 9 bits plus a stop and start bit. The difference between the two is, that in Mode 2 the serial port interrupt will not be activated unless the ninth data bit is a one; in Mode 3 the interrupt is activated whenever a byte is received. These two modes are commonly used for interprocessor communication.

Using XTAL1:

Mode 0: Baud Rate = $\frac{\text{XTAL1 frequency}}{4 \cdot (B + 1)}$; B \neq 0

Others: Baud Rate = $\frac{\text{XTAL1 frequency}}{64 \cdot (B + 1)}$

Using T2CLK:

Mode 0: Baud Rate = $\frac{\text{T2CLK frequency}}{B}$; B \neq 0

Others: Baud Rate = $\frac{\text{T2CLK frequency}}{16 \cdot B}$; B \neq 0

Note that B cannot equal 0, except when using XTAL1 in other than mode 0.

Figure 2-16. Baud Rate Formulas

Baud rates for all of the modes are controlled through the Baud Rate register. This is a byte wide register which is loaded sequentially with two bytes, and internally stores the value as a word. The least significant byte is loaded to the register followed by the most significant. The most significant bit of the baud value determines the clock source for the baud rate generator. If the bit is a one, the XTAL1 pin is used as the source, if it is a zero, the T2 CLK pin is used. The formulas shown in Figure 2-16 can be used to calculate the baud rates. The variable "B" is used to represent the least significant 15 bits of the value loaded into the baud rate register.

The baud rate register values for common baud rates are shown in Figure 2-17. These values can be used when XTAL1 is selected as the clock source for serial modes other than Mode 0. The percentage deviation from theoretical is listed to help assess the reliability of a given setup. In most cases a serial link will work if there is less than a 2.5% difference between the baud rates of the two systems. This is based on the assumption that 10 bits are transmitted per frame and the last bit of the frame must be valid for at least six-eighths of the bit time. If the two systems deviate from theoretical by 1.25% in opposite directions the maximum tolerance of 2.5% will be reached. Therefore, caution must be used when the baud rate deviation approaches 1.25% from theoretical. Note that an XTAL1 frequency of 11.0592 MHz can be used with the table values for 11 MHz to provide baud rates that have 0.0 percent deviation from theoretical. In most applications, however, the accuracy available when using an 11 MHz input frequency is sufficient.

Serial port Mode 1 is the easiest mode to use as there is little to worry about except initialization and loading and unloading SBUF, the Serial port BUFFER. If parity is enabled, (i.e., PEN = 1), 7 bits plus even parity are used instead of 8 data bits. The parity calculation is done in hardware for even parity. Modes 2 and 3 are similar to Mode 1, except that the ninth bit needs to be controlled and read. It is also not possible to enable parity in Mode 2. When parity is enabled in Mode 3 the ninth bit becomes the parity bit. If parity is not enabled, (i.e., PEN = 0), the TB8 bit controls the state of the ninth transmitted bit. This bit must be set prior to each transmission. On reception, if PEN = 0, the RB8 bit indicates the state of the ninth received bit. If parity is enabled, (i.e., PEN = 1), the same bit is called RPE (Receive Parity Error), and is used to indicate a parity error.

XTAL1 Frequency = 12.0 MHz		
Baud Rate	Baud Register Value	Percent Error
19.2K	8009H	+2.40
9600	8013H	+2.40
4800	8026H	-0.16
2400	804DH	-0.16
1200	809BH	-0.16
300	8270H	0.00
XTAL1 Frequency = 11.0 MHz		
19.2K	8008H	+0.54
9600	8011H	+0.54
4800	8023H	+0.54
2400	8047H	+0.54
1200	808EH	-0.16
300	823CH	+0.01
XTAL1 Frequency = 10.0 MHz		
19.2K	8007H	-1.70
9600	800FH	-1.70
4800	8020H	+1.38
2400	8040H	-0.16
1200	8081H	-0.16
300	8208H	+0.03

Figure 2-17. Baud Rate Values for 10, 11, 12 MHz

The software used to communicate between processors is simplified by making use of Modes 2 and 3. In a basic protocol the ninth bit is called the address bit. If it is set high then the information in that byte is either the address of one of the processors on the link, or a command for all the processors. If the bit is a zero, the byte contains information for the processor or processors previously addressed. In standby mode all processors wait in Mode 2 for a byte with the address bit set. When they receive that byte, the software determines if the next message is for them. The processor that is to

receive the message switches to Mode 3 and receives the information. Since this information is sent with the ninth bit set to zero, none of the processors set to Mode 2 will be interrupted. By using this scheme the overall CPU time required for the serial port is minimized.

A typical connection diagram for the multi-processor mode is shown in Figure 2-18. This type of communication can be used to connect peripherals to a desk top computer, the axis of a multi-axis machine, or any other group of microcontrollers jointly performing a task.

$$\text{Mode 2 Baud Rate} = \frac{\text{XTAL1 Frequency}}{8} \quad B = 0$$

$$\text{Other Baud Rate} = \frac{\text{XTAL1 Frequency}}{16 \cdot B} \quad B = 0$$

Note that B cannot equal 0, except when using XTAL1 in other than mode 0.

Figure 2-18. Baud Rate Formulas

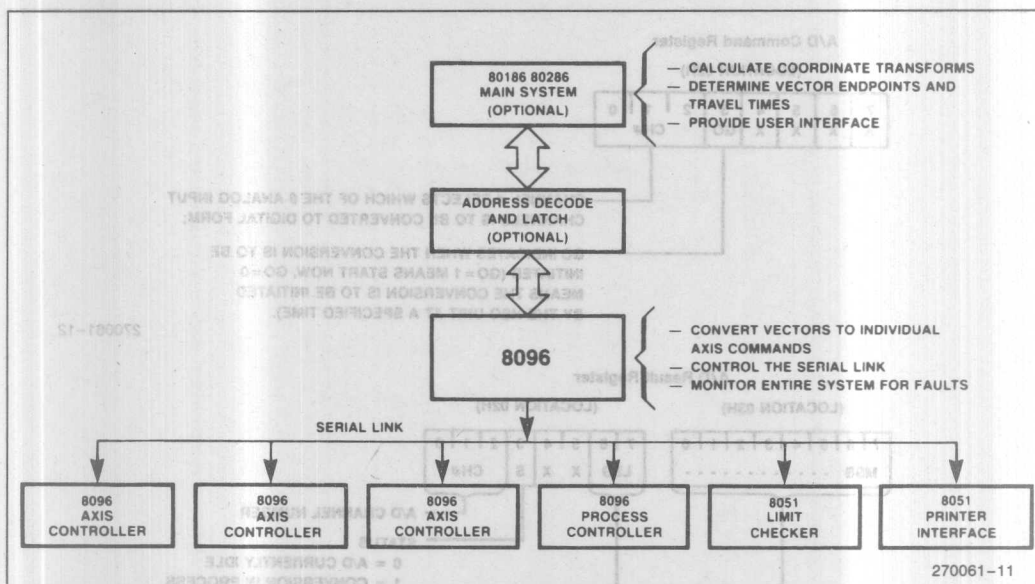


Figure 2-18. Multiprocessor Communication

Mode 0, the synchronous mode, is typically used for interfacing to shift registers for I/O expansion. The software to control this mode involves the REN (Receiver ENable) bit, the clearing of the RI bit, and writing to SBUF. To transmit to a shift register, REN is set to zero and SBUF is loaded with the information. The information will be sent and then the TI flag will be set. There are two ways to cause a reception to begin. The first is by causing a rising edge to occur on the REN bit, the second is by clearing RI with REN = 1. In either case, RI is set again when the received byte is available in SBUF.

2.3.5. A to D CONVERTER

Analog inputs are frequently required in a microcontroller application. The 8097 has a 10-bit A to D converter that can use any one of eight input channels. The conversions are done using the successive approximation method, and require 168 state times (42 microseconds with a 12 MHz clock.)

The results are guaranteed monotonic by design of the converter. This means that if the analog input voltage changes, even slightly, the digital value will either stay the same or change in the same direction as the analog

input. When doing process control algorithms, it is frequently the changes in inputs that are required, not the absolute accuracy of the value. For this reason, even if the absolute accuracy of a 10-bit converter is the same as that of an 8-bit converter, the 10-bit monotonic converter is much more useful.

Since most of the analog inputs which are monitored by a microcontroller change very slowly relative to the 42 microsecond conversion time, it is acceptable to use a capacitive filter on each input instead of a sample and hold. The 8097 does not have an internal sample and hold, so it is necessary to ensure that the input signal does not change during the conversion time. The input to the A/D must be between ANGND and VREF. ANGND must be within a few millivolts of VSS and VREF must be within a few tenths of a volt of VCC.

Using the A to D converter on the 8097 can be a very low software overhead task because of the interrupt and HSO unit structure. The A to D can be started by the HSO unit at a preset time. When the conversion is complete it is possible to generate an interrupt. By using these features the A to D can be run under complete interrupt control. The A to D can also be directly

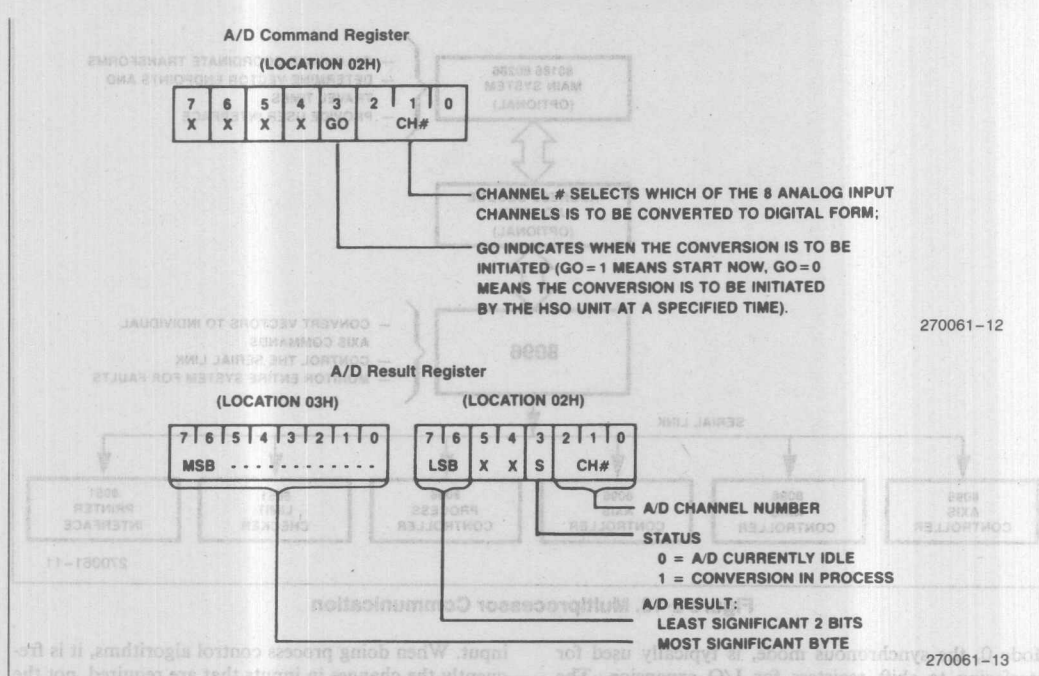


Figure 2-19. A to D Result/Command Register

controlled by software flags which are located in the AD_RESULT/AD_COMMAND Register, shown in Figure 2-19.

2.3.6. PWM REGISTER

Analog outputs are just as important as analog inputs when connecting to a piece of equipment. True digital to analog converters are difficult to make on a microprocessor because of all of the digital noise and the necessity of providing an on chip, relatively high current, rail to rail driver. They also take up a fair amount of silicon area which can be better used for other features. The A to D converter does use a D to A, but the currents involved are very small.

For many applications an analog output signal can be replaced by a Pulse Width Modulated (PWM) signal. This signal can be easily generated in hardware, and

takes up much less silicon area than a true D to A. The signal is a variable duty cycle, fixed frequency waveform that can be integrated to provide an approximation to an analog output. The frequency is fixed at a period of 64 microseconds for a 12 MHz clock speed. Controlling the PWM simply requires writing the desired duty cycle value (an 8-bit value) to the PWM Register. Some typical output waveforms that can be generated are shown in Figure 2-20.

Converting the PWM signal to an analog signal varies in difficulty, depending upon the requirements of the system. Some systems, such as motors or switching power supplies actually require a PWM signal, not a true analog one. For many other cases it is necessary only to amplify the signal so that it switches rail-to-rail, and then filter it. Switching rail-to-rail means that the output of the amplifier will be a reference value when the input is a logical one, and the output will

be zero when the input is a logical zero. The filter can be a simple RC network or an active filter. If a large amount of current is needed a buffer is also required. For low output currents, (less than 100 microamps or so), the circuit shown in Figure 2-21 can be used.

The RC network determines how quiet the output is, but the quieter the output, the slower it can change. The design of high accuracy voltage followers and active filters is beyond the scope of this paper, however many books on the subject are available.

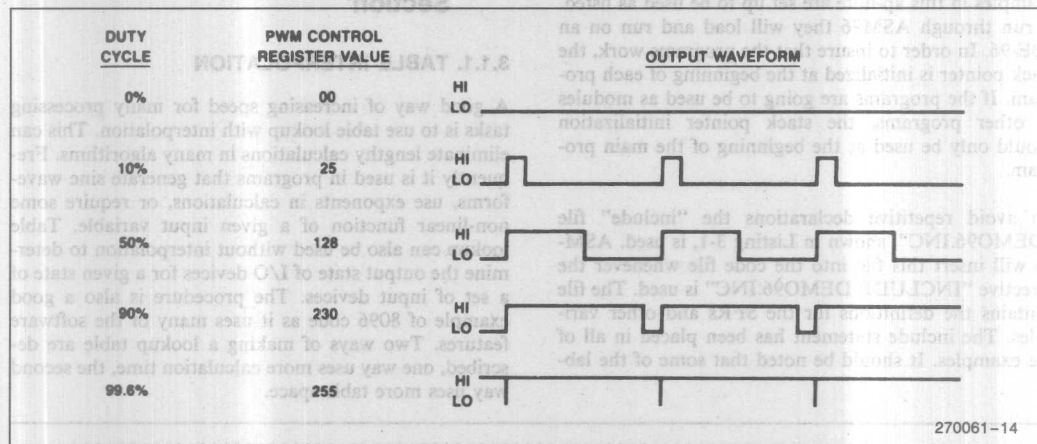


Figure 2-20. PWM Output Waveforms

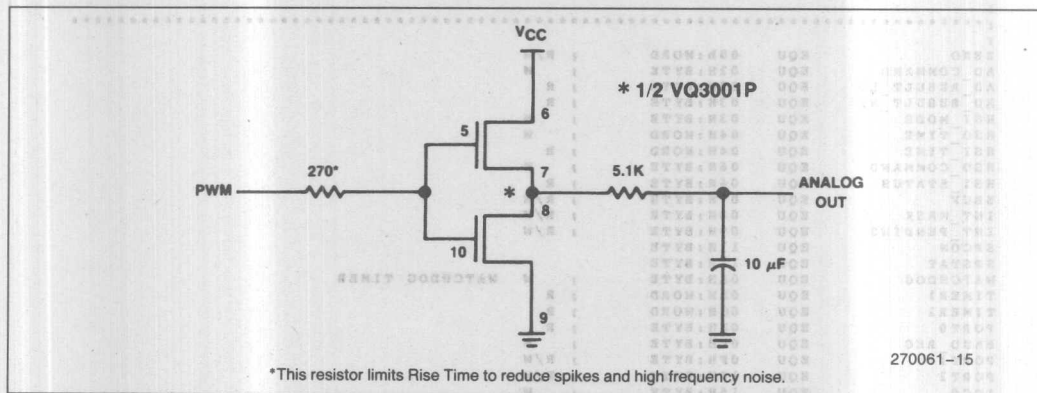


Figure 2-21. PWM to Analog Conversion Circuitry

3.0 BASIC SOFTWARE EXAMPLES

The examples in this section show how to use each I/O feature individually. Examples of using more than one feature at a time are described in section 4. All of the examples in this ap-note are set up to be used as listed. If run through ASM96 they will load and run on an SBE-96. In order to insure that the programs work, the stack pointer is initialized at the beginning of each program. If the programs are going to be used as modules of other programs, the stack pointer initialization should only be used at the beginning of the main program.

To avoid repetitive declarations the "include" file "DEMO96.INC", shown in Listing 3-1, is used. ASM-96 will insert this file into the code file whenever the directive "INCLUDE DEMO96.INC" is used. The file contains the definitions for the SFRs and other variables. The include statement has been placed in all of the examples. It should be noted that some of the lab-

els in this file are different from those in the file 8096.INC that is provided in the ASM-96 package.

3.1. Using the 8096's Processing Section

3.1.1. TABLE INTERPOLATION

A good way of increasing speed for many processing tasks is to use table lookup with interpolation. This can eliminate lengthy calculations in many algorithms. Frequently it is used in programs that generate sine waveforms, use exponents in calculations, or require some non-linear function of a given input variable. Table lookup can also be used without interpolation to determine the output state of I/O devices for a given state of a set of input devices. The procedure is also a good example of 8096 code as it uses many of the software features. Two ways of making a lookup table are described, one way uses more calculation time, the second way uses more table space.

DEMO96.INC - DEFINITION OF SYMBOLIC NAMES FOR THE I/O REGISTERS OF THE 8096			
ZERO	EQU	00h:WORD	; R/W
AD_COMMAND	EQU	02h:BYTE	; R/W
AD_RESULT_LO	EQU	02h:BYTE	; R
AD_RESULT_HI	EQU	03h:BYTE	; R
HSI_MODE	EQU	03h:BYTE	; R/W
HSO_TIME	EQU	04h:WORD	; R/W
HSI_TIME	EQU	04h:WORD	; R
HSO_COMMAND	EQU	06h:BYTE	; R/W
HSI_STATUS	EQU	06h:BYTE	; R
SBUF	EQU	07h:BYTE	; R/W
INT_MASK	EQU	08h:BYTE	; R/W
INT_PENDING	EQU	09h:BYTE	; R/W
SPCON	EQU	11h:BYTE	; R/W
SPSTAT	EQU	11h:BYTE	; R/W
WATCHDOG	EQU	0Ah:BYTE	; W WATCHDOG TIMER
TIMER1	EQU	0Ah:WORD	; R
TIMER2	EQU	0Ch:WORD	; R
PORT0	EQU	0Eh:BYTE	; R
BAUD_REG	EQU	0Eh:BYTE	; W
PORT1	EQU	0Fh:BYTE	; R/W
PORT2	EQU	10h:BYTE	; R/W
IOC0	EQU	15h:BYTE	; W
IOS0	EQU	15h:BYTE	; R
IOC1	EQU	16h:BYTE	; W
IOS1	EQU	16h:BYTE	; R
PWM_CONTROL	EQU	17h:BYTE	; W
SP	EQU	18h:WORD	; R/W STACK POINTER
RSEG at LCH			
AX:	DSW	1	
DX:	DSW	1	
BX:	DSW	1	
CX:	DSW	1	
AL	EQU	AX	: BYTE
AH	EQU	(AX+1)	: BYTE

Listing 3-1. Include File DEMO96.INC

If the input value was one of those listed then there would be no problem. Unfortunately the real world is never so kind. The input number will probably be 259 or something similar. If this is the case linear interpolation would provide a reasonable result. The formula is:

$$\text{Delta Out} = \frac{\text{UpperOutput}-\text{Lower Output}}{\text{Upper Input}-\text{Lower Input}} * (\text{Actual Input}-\text{Lower Input})$$

Actual Output = Lower Output + Delta Out
For the value of 259 the solution is:

$$\text{Delta Out} = \frac{900-400}{300-200} * (259-200) = \frac{500}{100} * 59 = 5 * 59 = 295$$

$$\text{Actual Output} = 400 + 295 = 695$$

is appropriate to limit the range and accuracy of the function to only what is needed. It is also advantageous to make the input step (Upper Input-Lower Input) equal to a power of 2. This allows the substitution of multiple right shifts for a divide operation, thus speeding up throughput. The 8096 allows multiple arithmetic right shifts with a single instruction providing a very fast divide if the divisor is a power of two.

For the purpose of an example, a program with a 12-bit output and an 8-bit input has been written. An input step of 16 ($2^{**}4$) was selected. To cover the input range 17 words are needed, $255/16 + 1$ word to handle values in the last 15 bytes of input range. Although only 12 bits are required for the output, the 16-bit architecture offers no penalty for using 16 instead of 12 bits.

The program for this example, shown in Listing 3-2, uses the definitions and equates from Listing 3-1, only the additional equates and definitions are shown in the code.

Input Value	Relative Table Address	Table Value
100	0001H	100
200	0002H	400
300	0003H	900
400	0004H	1600

```

$TITLE('INTER.APT: Interpolation routine 1')
$INCLUDE('DEMO96.INC') ; Include demo definitions

RSEG at 22H

IN_VAL: dsb 1 ; Actual Input Value
TABLE_LOW: dsb 1
TABLE_HIGH: dsb 1
IN_DIF: dsb 1 ; Upper Input - Lower Input
IN_DIFB: equ IN_DIF,byte 1
TAB_DIF: dsb 1 ; Upper Output - Lower Output
OUT: dsb 1
RESULT: dsb 1
OUT_DIF: dsb 1 ; Delta Out

CSEG at 2080H

LD SP, #100H

```

Listing 3-2. ASM-96 Code for Table Lookup Routine 1

```

look:  LDB     AL, IN_VAL      ; Load temp with Actual Value
       SHR     AL, #3         ; Divide the byte by 8
       AND     AL, #1111110B ; Insure AL is a word address
       ; This effectively divides AL by 2
       ; so AL = IN_VAL/16
       LDB     AX, AL         ; Load byte AL to word AX
       LD      TABLE_LOW, TABLE[AX] ; TABLE LOW is loaded with the value
       ; in the table at table location AX
       LD      TABLE_HIGH, (TABLE+2)[AX] ; TABLE HIGH is loaded with the
       ; value in the table at table
       ; location AX+2
       ; (The next value in the table)
       SUB     TAB_DIF, TABLE_HIGH, TABLE_LOW ; TAB_DIF=TABLE_HIGH-TABLE_LOW
       AND     IN_DIFB, IN_VAL, #0FH ; IN_DIFB=least significant 4 bits
       ; of IN_VAL
       LDB     IN_DIFB, IN_DIFB ; Load byte IN_DIFB to word IN_DIF
       MUL     OUT_DIF, IN_DIF, TAB_DIF ; Output difference =
       ; Input difference*Table difference
       ; Divide by 16 (2**4)
       SHRAL   OUT_DIF, #4
       ADD     OUT, OUT_DIF, TABLE_LOW ; Add output difference to output
       ; generated with truncated IN_VAL
       ; as input
       SHRA    OUT, #4 ; Round to 12-bit answer
       ADDC    OUT, zero ; Round up if Carry = 1
       no_inc: ST      OUT, RESULT ; Store OUT to RESULT
       BR      look ; Branch to "look:"
cseg    AT 2100H
table:  DCW     0000H, 2000H, 3400H, 4C00H ; A random function
       DCW     5D00H, 6A00H, 7200H, 7800H
       DCW     7B00H, 7D00H, 7600H, 6D00H
       DCW     5D00H, 4B00H, 3400H, 2200H
       DCW     1000H
END

```

270061-18

Listing 3-2. ASM-96 Code for Table Lookup Routine 1 (Continued)

If the function is known at the time of writing the software it is also possible to calculate in advance the change in the output function for a given change in the input. This method can save a divide and a few other instructions at the expense of doubling the size of the

lookup table. There are many applications where time is critical and code space is overly abundant. In these cases the code in Listing 3-3 will work to the same specifications as the previous example.

```

$TITLE('INTER2.APT: Interpolation routine 2')
; ; ; ; ; 8096 Assembly code for table lookup and interpolation
; ; ; ; ; Using tabled values in place of division
$INCLUDE('F1:DEMO96.INC') ; Include demo definitions

RSEG    at 24H

IN_VAL:  dsb     1 ; Actual Input Value
TABLE_LOW: dsb     1 ; Table value for function
TABLE_INC: dsb     1 ; Incremental change in function
IN_DIF:   dsb     1 ; Upper Input - Lower Input
IN_DIFB:  equ     IN_DIF ; byte
OUT:      dsb     1
RESULT:   dsb     1
OUT_DIF:  dsb     1 ; Delta Out

```

270061-19

Listing 3-3. ASM-96 Code For Table Lookup Routine 2

```

CSEG at 2080H
LD SP, #100H ; Initialize SP to top of reg. file
look: LDB AL, IN_VAL ; Load temp with Actual Value
SHRB AL, #3 ; Divide the byte by 8
ANDB AL, #11111110B ; Injure AL is a word address
; This effectively divides AL by 2
; so AL = IN_VAL/16
LDBEE AX, AL ; Load byte AL to word AX
LD TABLE_LOW, VAL_TABLE[AX] ; TABLE_LOW is loaded with the value
; in the value table at location AX
LD TABLE_INC, INC_TABLE[AX] ; TABLE_INC is loaded with the value
; in the increment table at
; location AX
ANDB IN_DIFB, IN_VAL, #0FH ; IN_DIFB=least significant 4 bits
; of IN_VAL
LDBEE IN_DIP, IN_DIFB ; Load byte IN_DIFB to word IN_DIP
MUL OUT_DIP, IN_DIP, TABLE_INC ; Output difference =
; Input_difference*Incremental_change
; JAV ADD OUT, OUT_DIP, TABLE_LOW ; Add output difference to output
; generated with truncated IN_VAL
; SHR OUT, #4 ; Round to 12-bit answer
; ADDC OUT, zero ; Round up if Carry = 1
no_inc: ST OUT, RESULT ; Store OUT to RESULT
BR look ; Branch to "look:"
cseg AT 2100H
val_table:
DCW 0000H, 2000H, 3400H, 4C00H ; A random function
DCW 5D00H, 6A00H, 7200H, 7800H
DCW 7B00H, 7D00H, 7600H, 6D00H
DCW 5D00H, 4B00H, 3400H, 2200H
DCW 1000H
inc_table:
DCW 0200H, 0140H, 0180H, 0110H ; Table of incremental
DCW 00D0H, 0080H, 0060H, 0030H ; differences
DCW 00020H, 0FF90H, 0FF70H, 0FF00H
DCW 0FE20H, 0FE90H, 0FEE0H, 0FEE0H
END

```

Listing 3-3. ASM-96 Code for Table Lookup Routine 2 (Continued)

By making use of the second lookup table, one word of RAM was saved and 16 state times. In most cases this time savings would not make much of a difference, but when pushing the processor to the limit, microseconds can make or break a design.

3.1.2. PL/M-96

Intel provides high level language support for most of its micro processors and microcontrollers in the form of PL/M. Specifically, PL/M refers to a family of languages, each similar in syntax, but specialized for the device for which it generates code. The PL/M syntax is similar to PL/I, and is easy to learn. PLM-96 is the version of PL/M used for the 8096. It is very code efficient as it was written specifically for the MCS-96 family. PLM-96 most closely resembles PLM-86, although it has bit and I/O functions similar to PLM-51. One line of PL/M-code can take the place of many

lines of assembly code. This is advantageous to the programmer, since code can usually be written at a set number of lines per hour, so the less lines of code that need to be written, the faster the task can be completed.

If the first example of interpolation is considered, the PLM-96 code would be written as shown in Listing 3-4. Note that version 1.0 of PLM-96 does not support 32-bit results of 16 by 16 multiplies, so the ASM-96 procedure "DMPY" is used. Procedure DMPY, shown in Listing 3-5, must be assembled and linked with the compiled PLM-96 program using RL-96, the relocater and linker. The command line to be used is:

```
RL96 PLMEX1.OBJ, DMPY.OBJ, PLM96.LIB &
to PLMOUT.OBJ ROM (2080H-3FFFH)
```

```

/* PLM-96 CODE FOR TABLE LOOK-UP AND INTERPOLATION */
PLMEX:    DO;

DECLARE IN_VAL    WORD    PUBLIC;
DECLARE TABLE_LOW    INTEGER    PUBLIC;
DECLARE TABLE_HIGH    INTEGER    PUBLIC;
DECLARE TABLE_DIF    INTEGER    PUBLIC;
DECLARE OUT        INTEGER    PUBLIC;
DECLARE RESULT    INTEGER    PUBLIC;
DECLARE OUT_DIF    LONGINT    PUBLIC;
DECLARE TEMP        WORD    PUBLIC;

DECLARE TABLE(17)    INTEGER DATA (1);
0000H, 2000H, 3400H, 4C00H, /* A random function */
5D00H, 6A00H, 7200H, 7800H,
7B00H, 7D00H, 7600H, 6D00H,
5D00H, 4B00H, 3400H, 2200H,
1000H);

DMPY:    PROCEDURE (A,B) LONGINT EXTERNAL;
DECLARE (A,B) INTEGER;
END DMPY;

LOOP:
TEMP=SHR(IN_VAL,4); /* TEMP is the most significant 4 bits of IN_VAL */
TABLE_LOW=TABLE(TEMP); /* If "TEMP" was replaced by "SHR(IN_VAL,4)" */
TABLE_HIGH=TABLE(TEMP+1); /* The code would work but the 8096 would */
/* do two shifts */

TABLE_DIF=TABLE_HIGH-TABLE_LOW;
OUT_DIF=DMPY(TABLE_DIF,SIGNED(IN_VAL AND 0FH)) /16;
OUT=SAR((TABLE_LOW+OUT_DIF),4); /* SAR performs an arithmetic right shift,
in this case 4 places are shifted */
IF CARRY=0 THEN RESULT=OUT; /* Using the hardware flags must be done */
ELSE RESULT=OUT+1; /* with care to ensure the flag is tested */
/* in the desired instruction sequence */

GOTO LOOP;

/* END OF PLM-96 CODE */
END;

```

270061-21

Listing 3-4. PLM-96 Code For Table Lookup Routine 1

\$TITLE('MULT.APT: 16*16 multiply procedure for PLM-96')

```

SP EQU 18H:word
EXTRN PLMREG:word
cseg
PUBLIC DMPY
; Multiply two integers and return a
; longint result in AX, DX registers
DMPY:    POP    PLMREG+4
POP    PLMREG
MUL    PLMREG,[SP]
; Load return address
; Load one operand
; Load second operand and increment SP
BR    [PLMREG+4]
; Return to PLM code
END

```

270061-22

Listing 3-5. 32-Bit Result Multiply Procedure For PLM-96

Using PLM, code requires less lines, is much faster to write, and easier to maintain, but may take slightly longer to run. For this example, the assembly code generated by the PLM-96 compiler takes 56.75 microseconds to run instead of 30.75 microseconds. If PLM-96 performed the 32-bit result multiply instead of using the ASM-96 routine the PLM code would take 41.5 microseconds to run. The actual code listings are shown in Appendix A.

3.2. Using the I/O Section

3.2.1. USING THE HSI UNIT

One of the most frequent uses of the HSI is to measure the time between events. This can be used for frequency determination in lab instruments, or speed/acceleration information when connected to pulse type encoders. The code in Listing 3-6 can be used to determine the high and low times of the signals on two lines. This code can be easily expanded to 4 lines and can also be modified to work as an interrupt routine.

Frequently it is also desired to keep track of the number of events which have occurred, as well as how often they are occurring. By using a software counter this feature can be added to the above code. This code depends on the software responding to the change in line state before the line changes again. If this cannot be guaranteed then it may be necessary to use 2 HSI lines for each incoming line. In this case one HSI line would look for falling edges while the other looks for rising edges. The code in Listing 3-7 includes both the counter feature and the edge detect feature.

The uses for this type of routine are almost endless. In instrumentation it can be used to determine frequency on input lines, or perhaps baud rate for a self adjusting serial port. Section 4.2 contains an example of making a software serial port using the HSI unit. Interfacing to some form of mechanically generated position information is a very frequent use of the HSI. The applications in this category include motor control, precise positioning (print heads, disk drives, etc.), engine control and

```

$TITLE('PULSE.APT: Measuring pulses using the HSI unit')
$INCLUDE(Demo96.INC)

rseg at 26H
HIGH TIME: dsw 1
LOW TIME: dsw 1
PERIOD: dsw 1
HI_EDGE: dsw 1
LO_EDGE: dsw 1

cseg at 2080H
LD SP, #100H
LDB IOC0, #00000001b ; Enable HSI 0
LDB HSI_MODE, #00001111b ; HSI 0 look for either edge

wait: ADD PERIOD, HIGH TIME, LOW TIME
JBS IOS1, 6, contin ; If FIFO is full
JBC IOS1, 7, wait ; Wait while no pulse is entered

contin: LDB AL, HSI_STATUS ; Load status; Note that reading
; HSI_TIME clears HSI_STATUS
LD BX, HSI_TIME ; Load the HSI_TIME
JBS AL, 1, hai_hi ; Jump if HSI.0 is high

hai_lo: ST BX, LO_EDGE
SUB HIGH TIME, LO_EDGE, HI_EDGE
wait

hai_hi: ST BX, HI_EDGE
SUB LOW TIME, HI_EDGE, LO_EDGE
wait

END

```

Listing 3-6. Measuring Pulses Using The HSI Unit

transmission control. The HSI unit is used extensively in the example in section 4.3.

3.2.2. USING THE HSO UNIT

Although the HSO has many uses, the best example is that of a multiple PWM output. This program, shown in Listing 3-8, is simple enough to be easily understood, yet it shows how to use the HSO for a task which can be complex. In order for this program to operate, another program needs to set up the on and off time variables for each line. The program also requires that a

HSO line not change so quickly that it changes twice between consecutive reads of I/O Status Register 0, (IOS0).

A very eye catching example can be made by having the program output waveforms that vary over time. The driver routine in Listing 3-10 can be linked to the above program to provide this function. Linking is accomplished using RL96, the relocatable linker for the 8096. Information for using RL96 can be found in the "MCS-96 Utilities Users Guide", listed in the bibliography. In order for the program to link, the register dec-

```

$TITLE ('ENHSI.APT: ENHANCED HSI PULSE ROUTINE')
$INCLUDE (DEMO96.INC)
RSEG AT 28H

TIME: DSW 1
LAST_RISE: DSW 1
LAST_FALL: DSW 1
HSI_SO: DSB 1
IOS1_BAK: DSB 1
PERIOD: DSW 1
LOW_TIME: DSW 1
HIGH_TIME: DSW 1
COUNT: DSW 1

cseg at 2080H
init: LD SP, #100H
LDB IOC1, #00100101B ; Disable HSO.4, HSO.5, HSI_INT-first,
; Enable PWM, TKD, TIMER1_OVERFLOW_INT
LDB HSI_MODE, #10011001B ; set hsi.1 -; hsi.0 +
LDB IOC0, #00000111B ; Enable hsi 0,1
; T2 CLOCK-T2CLK, T2RST-T2RST
; Clear timer2

wait: ANDB IOS1_BAK, #01111111B ; Clear IOS1_BAK.7
ORB IOS1_BAK, IOS1 ; Store into temp to avoid clearing
; other flags which may be needed
JBC IOS1_BAK, 7, wait ; If hsi is not triggered then
; jump to wait

ANDB HSI_SO, HSI_STATUS, #01010101B
LD TIME, HSI_TIME

JBS HSI_SO, 0, a_rise
JBS HSI_SO, 2, a_fall
BR no_cnt

a_rise: SUB LOW_TIME, TIME, LAST_FALL
SUB PERIOD, TIME, LAST_RISE
LD LAST_RISE, TIME
BR increment

a_fall: SUB HIGH_TIME, TIME, LAST_RISE
SUB PERIOD, TIME, LAST_FALL
LD LAST_FALL, TIME

increment: INC COUNT

no_cnt: BR wait

END

```

Listing 3-7. Enhanced HSI Pulse Measurement Routine

270061-24

laration section (i.e., the section between "RSEG" and "CSEG") in Listing 3-8 must be changed to that in Listing 3-9.

The driver routine simply changes the duty cycle of the waveform and sets the second HSO output to a fre-

quency twice that of the first one. A slightly different driver routine could easily be the basis for a switching power supply or a variable frequency/variable voltage motor driver. The listing of the driver routine is shown in Listing 3-10.

```

; NOTE: Use this file to replace the declaration section of
; the HSO PWM program from "INCLUDE(DEMO96.INC)" through
; the line prior to the label "wait". Also change the last
; branch in the program to a "RET".
RSEG

D_STAT:      DSB      1
extrn  HSO_ON_0 :word , HSO_OFF_0 :word
extrn  HSO_ON_1 :word , HSO_OFF_1 :word
extrn  HSO_TIME :word , HSO_COMMAND :byte
extrn  TIMER1   :word , IOS0    :byte
extrn  SP       :word

public  OLD_STAT
OLD_STAT:  dsb      1
NEW_STAT:  dsb      1

cseg
PUBLIC  wait

```

270061-26

Listing 3-9. Changes to Declarations for HSO Routine

```

$TITLE('HSODRV.APT: Driver module for HSO PWM program')
HSODRV MODULE MAIN, STACKSIZE(8)

PUBLIC  HSO_ON_0 , HSO_OFF_0
PUBLIC  HSO_ON_1 , HSO_OFF_1
PUBLIC  HSO_TIME , HSO_COMMAND
PUBLIC  SP , TIMER1 , IOS0

$INCLUDE(DEMO96.INC)

rseg at 28H
EXTRN  OLD_STAT      ,byte

HSO_ON_0:  dsb      1
HSO_OFF_0: dsb      1
HSO_ON_1:  dsb      1
HSO_OFF_1: dsb      1
count:    dsb      1

cseg at 2080H
EXTRN  wait          ,entry

strt:  DI      SP, #100H
      ANDB    OLD_STAT, IOS0, #0FH
      XORB    OLD_STAT, #0FH

initial: LD     CX, #0100H

loop:  LD     AX, #1000H
      SUB    BX, AX, CX
      LD     AX, CX
      ST     AX, HSO_ON_0
      ST     BX, HSO_OFF_0

```

270061-27

Listing 3-10. Driver Module for HSO PWM Program

that the HSO unit can be used to initiate the desired tasks at the appropriate tooth count. The interrupt routine initiated by HSI.0 can be used to perform any software task required every revolution. In this system, the overhead which would normally require extensive software has been done with the hardware on the 8096, thus making more software time available for control programs.

3.2.3. USING THE SERIAL PORT IN MODE 1

Mode 1 of the serial port supports the basic asynchronous 8-bit protocol and is used to interface to most CRTs and printers. The example in Listing 3-11 shows a simple routine which receives a character and then

transmits the same character. The code is set up so that minor modifications could make it run on an interrupt basis. Note that it is necessary to set up some flags as initial conditions to get the routine to run properly. If it was desired to send 7 bits of data plus parity instead of 8 bits of data the PEN bit would be set to a one. Inter-processor communication, as described in section 2.3.4, can be set up by simply adding code to change RB8 and the port mode to the listing below. The hardware shown in Figure 3-2 can be used to convert the logic level output of the 8096 to ± 12 or 15 volt levels to connect to a CRT. This circuit has been found to work with most RS-232 devices, although it does not conform to strict RS-232 specifications. If true RS-232 conformance is required then any standard RS-232 driver can be used.

```

$TITLE('SP.APT: SERIAL PORT DEMO PROGRAM')
$INCLUDE(DEMO96.INC)

rseg at 28H
    CHR:   ddb 1
    SPTMP: ddb 1
    TEMPO: ddb 1
    TEMPI: ddb 1
    RCV FLAG: ddb 1

cseg at 200CH
    DCW ser_port_int

cseg at 2080H
    LD SP, #100H
    LDB IOCL, #00100000B
    ; Baud rate = input frequency / (64*baud_val)
    ; baud_val = (input frequency/64) / baud_rate

    baud_val equ 39
    BAUD_HIGH equ ((baud_val-1)/256) OR 80H
    BAUD_LOW equ (baud_val-1) MOD 256
    ; Set MSB to 1

    LDB BAUD_REG, #BAUD_LOW
    LDB BAUD_REG, #BAUD_HIGH
    LDB SPCON, #01001001B
    ; Enable receiver, Mode 1
    ; The serial port is now initialized

    STB SBUF, CHR
    LDB TEMPO, #00100000B
    ; Clear serial port
    ; Set TI-temp

    LDB INT_MASK, #01000000B
    ; Enable Serial Port Interrupt

loop: BR loop
    ; Wait for serial port interrupt

ser_port_int:
    PUSHF
rd_again:
    LDB SPTMP, SPSTAT
    ORB TEMPO, SPTMP
    ANDB SPTMP, #01100000B
    JNE rd_again
    ; This section of code can be replaced
    ; with "ORB TEMPO, SP_STAT" when the
    ; serial port TI and RI bugs are fixed
    ; Repeat until TI and RI are properly cleared

```

270061-30

Listing 3-11. Using the Serial Port in Mode 1

```

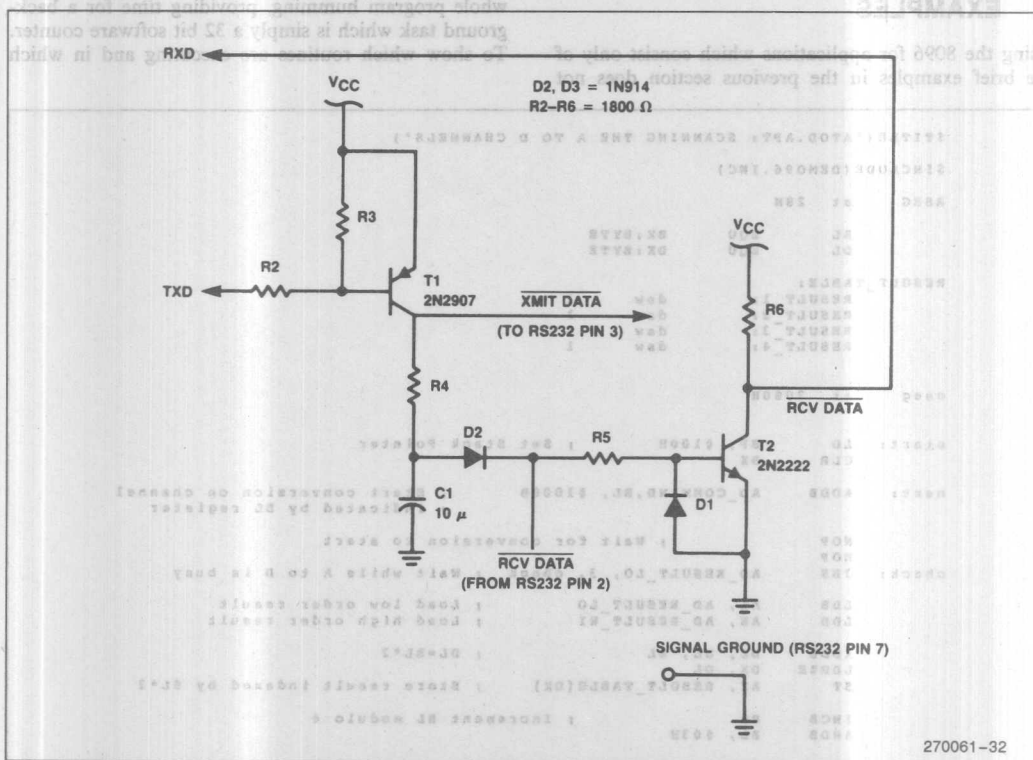
get_byte:
    JBC     TEMP0, 6, put_byte
    STB     SBUF, CHR
    ANDB    TEMP0, #10111111B
    LDB     RCV_FLAG, #0FFH
    put_byte:
    JBC     RCV_FLAG, 0, continue
    JBC     TEMP0, 5, continue
    LDB     SBUF, CHR
    ANDB    TEMP0, #10111111B
    ANDB    CHR, #01111111B
    CMPB    CHR, #00DH
    JNE     clr_rcv
    LDB     CHR, #0AH
    BR      continue

clr_rcv:
    CLRB    RCV_FLAG

continue:
    POPF
    RET
    END

```

Listing 3-11. Using the Serial Port in Mode 1 (Continued)



3.2.4. USING THE A TO D

The code in Listing 3-12 makes use of the software flags to implement a non-interrupt driven routine which scans A to D channels 0 through 3 and stores them as words in RAM. An interrupt driven routine is shown in section 4.1. When using the A to D it is important to always read the value using the byte read commands, and to give the converter 8 state times to start converting before reading the status bit.

Since there is no sample and hold on the A to D converter it may be desirable to use an RC filter on each input. A 100 Ω resistor in series with a 0.22 μ f capacitor to ground has been used successfully in the lab. This circuit gives a time constant of around 22 microseconds which should be long enough to get rid of most noise, without overly slowing the A to D response time.

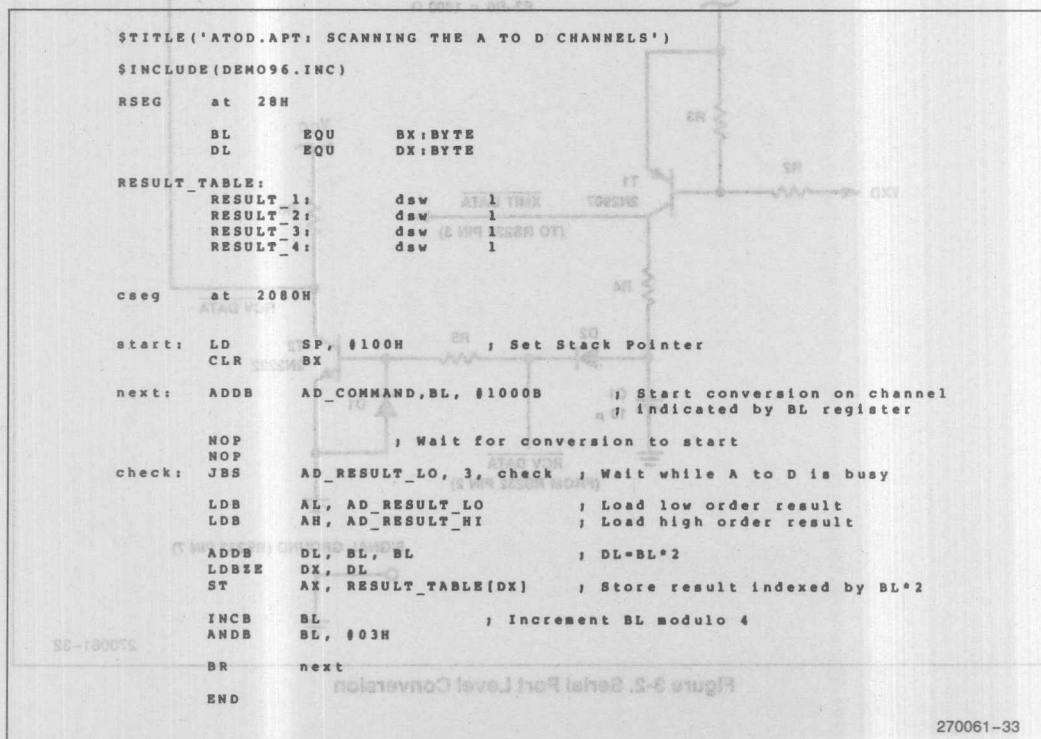
4.0 ADVANCED SOFTWARE EXAMPLES

Using the 8096 for applications which consist only of the brief examples in the previous section, does not

really make use of its full capabilities. The following examples use some of the code blocks from the previous section to show how several I/O features can be used together to accomplish a practical task. Three examples will be shown. The first is simply a combination of several of the section 3 examples run under an interrupt system. Next, a software serial port using the HSIO unit is described. The concluding example is one of interfacing the HSI unit to an optical encoder to control a motor.

4.1. Simultaneous I/O Routines under Interrupt Control

A four channel analog to PWM converter can easily be made using the 8096. In the example in Listing 4 analog channels are read and 3 PWM waveforms are generated on the HSO lines and one on the PWM pin. Each analog channel is used to set the duty cycle of its associated output pin. The interrupt system keeps the whole program humming, providing time for a background task which is simply a 32 bit software counter. To show which routines are executing and in which



Listing 3-12. Scanning the A to D Channels


```

LDB HSO_COMMAND, #00110110B ; Set HSO for timer1, set pin 0,1
LD HSO_TIME, NXT_ON_T ; with interrupt
NOP
NOP
LDB HSO_COMMAND, #00100010B ; Set HSO for timer1, set pin 2
ADD HSO_TIME, NXT_ON_T ; without interrupt

ORB LAST_LOAD, #00000111B ; Last loaded value was set all pins
LDB INT_MASK, #00001010B ; Enable HSO and A/D interrupts
LDB INT_PENDING, #00001010B ; Fake an A/D and HSO interrupt
EI

loop: ORB Port1, #00000001B ; set P1.0
ADD COUNT, #01
ADDC COUNT, 1, zero
ANDB Port1, #11111110B ; clear P1.0
BR loop

```

270061-35

Listing 4-1a. Initializing the A to D to PWM program (Continued)

```

; HSO EXECUTED INTERRUPT
; HSO EXECUTED INTERRUPT
; HSO EXECUTED INTERRUPT

HSO_exec int:
PUSHF
ORB Port1, #00000010B ; Set p1.1

SUB TMP, TIMER1, NXT_ON_T
CMP TMP, ZERO
JLT set_off_times

set_on_times:
ADD NXT_ON_T, HSO_PER
LDB HSO_COMMAND, #00110110B ; Set HSO for timer1, set pin 0,1
LD HSO_TIME, NXT_ON_T
NOP
NOP
LDB HSO_COMMAND, #00100010B ; Set HSO for timer1, set pin 2
LD HSO_TIME, NXT_ON_T
ORB LAST_LOAD, #00000111B ; Last loaded value was all ones
LDB PWM_CONTROL, PWM_TIME_1 ; Now is as good a time as any
BR check_done ; to update the PWM reg

set_off_times:
JBC LAST_LOAD, 0, check_done

ADD NXT_OFF_0, NXT_ON_T, HSO_ON_0
LDB HSO_COMMAND, #00010000B ; Set HSO for timer1, clear pin 0
LD HSO_TIME, NXT_OFF_0

NOP
ADD NXT_OFF_1, NXT_ON_T, HSO_ON_1
LDB HSO_COMMAND, #00010001B ; Set HSO for timer1, clear pin 1
LD HSO_TIME, NXT_OFF_1

NOP
ADD NXT_OFF_2, NXT_ON_T, HSO_ON_2
LDB HSO_COMMAND, #00010010B ; Set HSO for timer1, clear pin 2
LD HSO_TIME, NXT_OFF_2

ANDB LAST_LOAD, #11111000B ; Last loaded value was all 0s

check_done:
ANDB Port1, #11111101B ; Clear P1.1
POPF
RET

```

270061-36

Listing 4-1b. Interrupt Driven HSO Routine

The table also shows the declarations for the commands issued to the HSO unit. In this example HSI.2 is used for receive data and HSO.5 is used for transmit data, although other HSI and HSO lines could have been used.

The interface routines are shown in Listing 4-3. Data is passed to the port by pushing the eight-bit character into the stack and calling `char_out`, which waits for any in-process transmission to complete and stores the character into the variable `serial_out`. As the data is

stored the START and STOP bits are added to the data bits. The routine `char_in` is called when the application software requires a character from the port. The data is returned in the `ax` register in conformance to PLM 96 calling conventions. The routine `csts` can be called to determine if a character is available at the port before calling `char_in`. (If no character is available `char_in` will wait indefinitely).

The initialization routine is shown in Listing 4-4. This routine is called with the required baud rate in the

```

;
; char_out:
; Output character to the software serial port
;
;       pop     cx             ; the return address
;       pop     bx             ; the character for output
;       ldb     (bx+1),001h     ; add the start and stop bits
;       add     bx,bx           ; to the char and leave as 16 bit
;
; wait_for_xmit:
;       cmp     serial_out,0     ; wait for serial_out=0 (it will be cleared by
;       bne     wait_for_xmit    ; the hso interrupt process)
;       st      bx,serial_out    ; put the formatted character in serial_out
;       br      [cx]            ; return to caller
;
; csts:
; Returns "true" (ax<>0) if char_in has a character.
;
;       clr     ax
;       bbc     rcve_state,0,csts_exit
;       inc     ax
;
; csts_exit:
;       ret
;
; char_in:
; Get a character from the software serial port
;
;       ; wait for character ready
;       bbc     rcve_state,0,char_in
;       pushf
;       andb    rcve_state,#not(rxdy)
;       ldb     al,rcve_buf
;       popf
;       ret
;       ; leave the critical region

```

Listing 4-3. Software Serial Port Interface Routines

```

;
; setup_serial_port:
; Called on system reset to initiate the software serial port.
;
;       pop     cx             ; the return address
;       pop     bx             ; the baud rate (in decimal)
;       ld      dx,00007h      ; dx:ax:=500,000 (assumes 12 Mhz crystal)
;       ld      ax,00A120h
;       divu    ax,bx           ; calculate the baud count (500,000/baudrate)
;       st      ax,baud_count
;       st      0,serial_out    ; clear serial_out
;       ioclr   0,01100000b     ; Enable HSO.5 and Txd
;       bbs     ios0,6,6       ; Wait for room in the HSO CAM
;                               ; and issue a MARK command.
;       add     txd_time,timer1,20
;       hso_command,#mark_command
;       ld      hso_time,txd_time
;       clrb    rcve_buf
;       clrb    rcve_reg
;       clrb    rcve_state
;       call    init_receive    ; setup to detect a start bit
;       br      [cx]           ; return

```

Listing 4-4. Software Serial Port Initialization Routine

stack; it calculates the bit time from the baud rate and stores it in the variable *baud_count* in units of TIMER1 ticks. An HSO command is issued which will initiate the transmit process and then the remainder of the variables owned by the port are initialized. The routine *init_receive* is called to setup the HSI unit to look for the leading edge of the START bit.

The transmit process is shown in Listing 4-5. The HSO unit is used to generate an output command to the transmit pin once per bit time. If the *serial_out* register is zero a MARK (idle condition) is output. If the *serial_out* register contains data then the least sig-

nificant bit is output and the register shifted right one place. The framing information (START and STOP bits) are appended to the actual data by the interface routines. Note that this routine will be executed once per bit time whether or not data is being transmitted. It would be possible to use this routine for additional low resolution timing functions with minimal overhead.

The receive process consists of an initialization routine and two interrupt service routines, *hsi_isr* and *software_timer_isr*. The listings of these routines are shown in Listings 4-6a, 4-6b, and 4-6c respectively. The

```

;
; hso_isr:
; Fields the hso interrupts and performs the serialization of the data.
; Note: this routine would be incorporated into the hso-service strategy for an
; actual system.
;
; at 2006h
; hso_isr: Set up vector
;
; cseg
; dcb
;
; cseg
; pushf
; add
; cmp
; be
; shr
; bc
; send_space:
; ldb
; ld
; br
; send_mark:
; ldb
; ld
;
; hso_isr_exit:
; popf
; ret
;
; reject

```

Listing 4-5. Software Serial Port Transmit Process

Listing 4-6. Receive Process

```

;
; init_receive:
; Called to prepare the serial input process to find the leading edge of
; a start bit.
;
; ldb
; ldb
; flush_fifo:
; orb
; bbc
; al, hsi_status
; ld
; andb
; br
; flush_fifo_done:
; ldb
; ret

```

Listing 4-6a. Software Serial Port Receive Initialization

```

; Fields interrupts from the HSI unit, used to detect the leading edge
; of the START bit
; Note: this routine would be incorporated into the HSI strategy of an actual
; system.
;
; Setup the interrupt vector
;
; Wait for room in HSO holding reg
; send out sample command in 1/2
; bit time
; disconnect hsi.2 from change detector
;
; exit_hsi:
; pop ax
; popf
; ret

```

270061-44

Listing 4-6b. Software Serial Port Start Bit Detect

```

; Fields the software timer interrupt, used to deserialize the incoming data.
; Note: this routine would be incorporated into the software timer strategy
; in an actual system.
;
; Setup vector
;
; Clear bit 0
; All bits except rxrdy and overrun=0
;
; init_receive
;
; set receive in progress flag
;
; set the new data bit
;
; increment bit count
;
; DEBUG ONLY
; Clear all but ready and overrun bits
;
; wait for holding reg empty
;

```

270061-45

Listing 4-6c. Software Serial Port Data Reception

start is detected by the *hsi_isr* which schedules a software timer interrupt in one-half of a bit time. This first sample is used to verify that the START bit has not ended prematurely (a protection against a noisy line). The software timer service routine uses the variable *rcve_state* to determine whether it should check for a valid START bit, deserialize data, or check for a valid STOP bit. When a complete character has been received it is moved to the receive buffer and *init_receive* is called to set up the receive process for the next character. This routine is also called when an error (e.g., invalid START bit) is detected.

Appendix C contains the complete listing of the routines and the simple loop which was used to initialize them and verify their operation. The test was run for several hours at 9600 baud with no apparent malfunction of the port.

4.3. Interfacing an Optical Encoder to the HSI Unit

Optical encoders are among one of the more popular devices used to determine position of rotating equipment. These devices output two pulse trains with edges that occur from 2 to 4000 times a revolution.

Frequently there is a third line which generates one pulse per revolution for indexing purposes. Figure 4-2 shows a six line encoder and typical waveforms. As can be seen, the two waveforms provide the ability to determine both position and direction. Since a microcontroller can perform real time calculations it is possible to determine velocity and acceleration from the position and time information.

Interfacing to the encoder can be an interesting problem, as it requires connecting mechanically generated electrical signals to the HSI unit. The problems arise because it is difficult to obtain the exact nature of the signals under all conditions.

The equipment used in the lab was a Pittman 9400 series gearmotor with a 600 line optical encoder from Vernitech. The encoder has to be carefully attached to the shaft to minimize any runout or endplay. Fortunately, Pitmann has started marketing their motors with ball bearings and optical encoders already installed. It is recommended that the encoder be mounted to the motor using the exact specifications of the encoder manufacturer and/or a good machine shop.

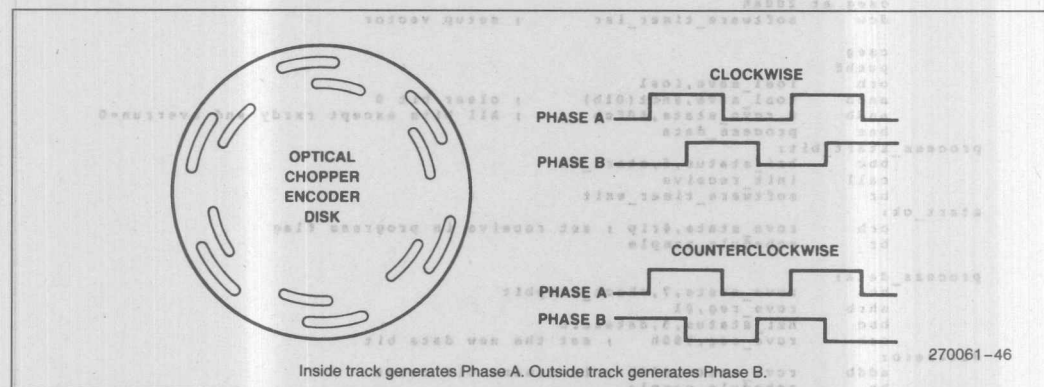


Figure 4-2. Optical Encoder and Waveforms

Digital filtering external to the 8096 is used on the encoder signals. The idealized signals coming from the encoder and after the digital filter are shown in Figure 4-3. The circuitry connecting the encoder to the 8096 requires only two chips. A one-shot constructed of XOR gates generates pulses on each edge of each signal. The pulses generated by Phase A are used to clock the signal from Phase B and vice versa. The hardware is shown in Figure 4-4. CMOS parts are used to reduce loading on the encoder so that buffers are not needed. Note that T2CLK is clocked on both edges of both filtered phases.

By using this method repetitive edges on a single phase without an edge on the other phase will not be passed on to the 8096. Repetitive edges on a phase can occur when the motor is stopped and vibrates or when it is changing direction. The digital filtering technique causes a little more delay in the signal at slow speeds than an analog filter would, but the simplicity trade off is worthwhile. The net effect of digital filtering is losing the ability to determine the first edge after a direction change. This does not affect the count since the first edge in both directions is lost.

If it is desired to determine when each edge occurs before filtering, the encoder outputs can be attached directly to the 8096. As these would be input signals, Port 0 is the most likely choice for connection. It would not be required to connect these lines to the HSI unit, as the information on them would only be needed when the motor is going very slowly.

The motor is driven using the PWM output pin for power control and a port pin for direction control. The 8096 drives a 7438 which drives 2 opto-isolators. These in turn drive two VFETs. A MOV (Metal Oxide Varistor, a type of transient absorber) is used to protect the VFETs, and a capacitor filters the PWM to get the best motor performance. Figure 4-5 shows the driver circuitry. To avoid noise getting into the 8096 system, the ± 15 volt power supply is isolated from the 8096 logic power supply.

This is the extent of the external circuitry required for this example. All of the counting and direction detection are done by the 8096. There are two sections to the example: driving the motor and interfacing to the encoder. The motor driver uses proportional control with

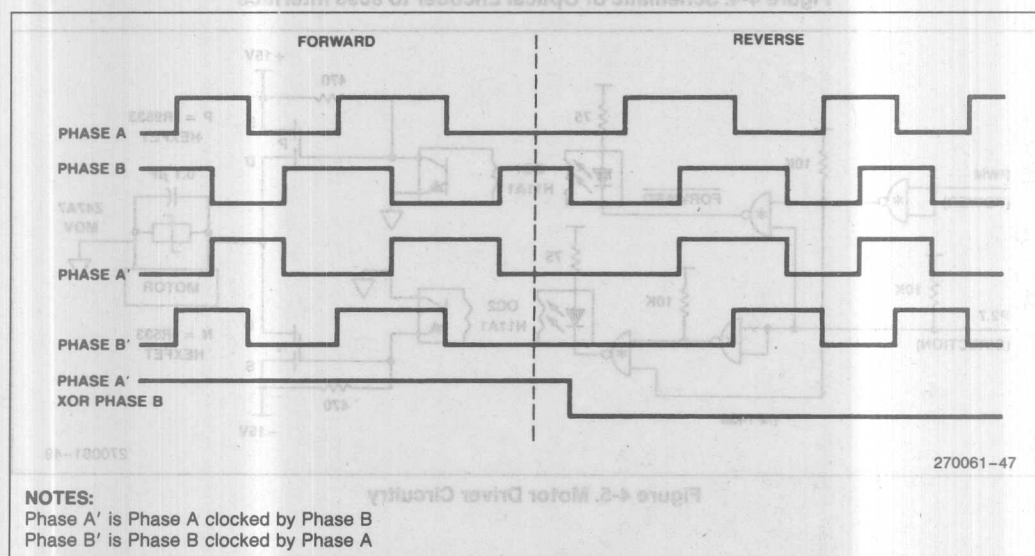


Figure 4-3. Filtered Encoder Waveforms

some modifications and a braking algorithm. Since the main point of this example is I/O interfacing, the motor driver will be briefly described at the end of this section.

In order to interface to the encoder it is necessary to know the types of waveforms that can be expected. The motor was accelerated and decelerated many times using different maximum voltages. It was found that the

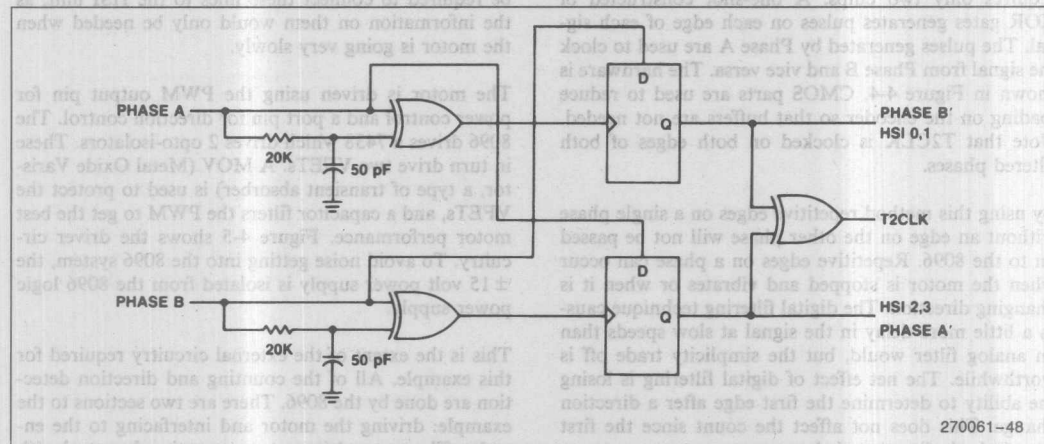


Figure 4-4. Schematic of Optical Encoder to 8096 Interface

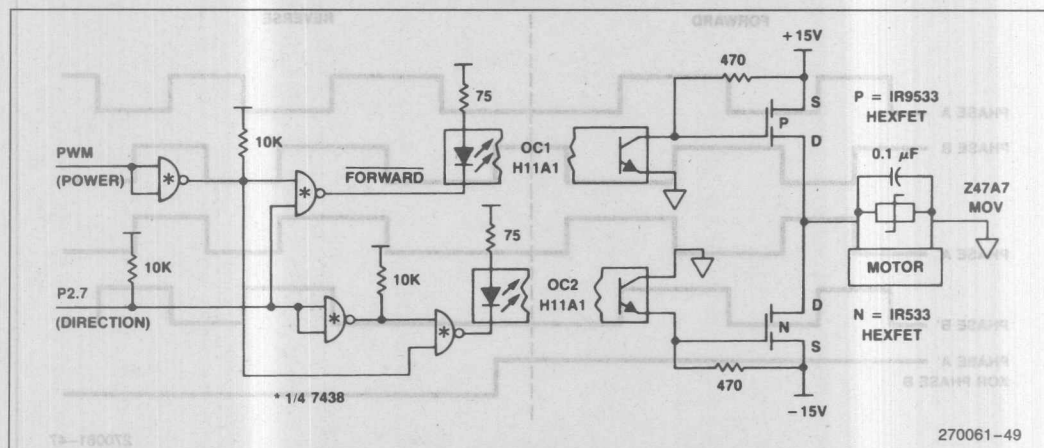


Figure 4-5. Motor Driver Circuitry


```

in_mode2:
    sub    delta_p,timer_2,tmr2_old
    ld     tmr2_old,timer_2
    jbc    direct,0,in_rev
in_fwd:   add    position,delta_p
    addc   position+2,zero
    bnc    chk_mode
in_rev:   sub    position,delta_p
    subc   position+2,zero

chk_mode:
    sub    tmpl,timer_2,old_t2
    cmp    tmpl,#5
    jgt    end_sw0
    set_model:
        andb    port1,$11111101B
        orb     port1,$000000001B
        ldb     ioc0,$000000101B
        ld      zero,HSI_TIME
        sub     last1_time,timer1,min_hsl1
        ; set up so (time-last2_time)>min_hsl1 on next HSI
        clr_hsl:
            ld      zero,HSI_TIME
            andb    ioc1_bak,$01111111B
            orb     ioc1_bak,ioc1
            jbs     ioc1_bak,7,clr_hsl
        end_sw0:
            ld      old_t2,timer_2
            andb    port1,$11011111B
            POPF
            ret

```

Listing 4-7. Motor Control HSO.0 Timer Routine (Continued)

If the pulse rate is slow enough to go to mode 1, the transition is made by enabling HSI.0 and HSI.1. Both of these lines are connected to the same encoder line, with HSI.0 looking for rising edges and HSI.1 looking for falling edges. The *HSI_TIME* register is read to speed up clearing the HSI FIFO and the *LAST1_TIME* value is set up so the mode 1 routine does not immediately put the program into another mode. The HSI FIFO is then cleared, the Timer 2 value used throughout this routine is saved, and the routine returns.

This routine still runs in modes 0 and 1, but in an abbreviated form. The section of code starting with the label *in_model* checks to see if the pulses are coming in so slowly that both HSI lines can be checked. If this is the case then all of the HSIs are enabled and the program returns. This routine is the secondary method for going from mode 1 to mode 0, the primary method is by checking the time between edges during the HSI routine, which will be described later.

The HSO routine will enable mode 0 from mode 1 if two edges are not received every 260 microseconds. The primary method, (under the HSI routine), can only

motor would decelerate smoothly until the time between edges was around 100 microseconds. At this point the motor would suddenly stop and reverse. The latter case is the one that was most problematic.

After a brief overview, each section of the program will be described separately, with the complete listing included in the Appendix D. In order to make debugging easier, as well as to provide insight into how the program is working, two port I/O routines are used. The program is in an under which is actually a loop, HSI. The main program sections are: Main loop, HSI interrupt, and software timer handling. The main loop section is described separately as initialization, timer handling, and software timer handling. The HSI interrupt routine is described separately as initialization, timer handling, and software timer handling. The software timer handling routine is described separately as initialization, timer handling, and software timer handling. The HSI interrupt routine is described separately as initialization, timer handling, and software timer handling. The software timer handling routine is described separately as initialization, timer handling, and software timer handling.

270061-51

enable mode 0 after an edge is received. This could cause a problem if the last 2 edges on Phase A before the encoder stops were too close to enable mode 0. If this happened, mode 0 would not be enabled until after the encoder started again, resulting in missed edges on Phase B. Using the HSO routine to switch from mode 1 to mode 0 eliminates this problem.

Figure 4-6 shows a state diagram of how the mode switching is done. As can be seen, there are two sources for most of the mode decisions. This helps avoid problems such as the one mentioned above.

When either Mode 1 or Mode 0 is enabled the HSI interrupt routine performs the counting of edges, while the HSO routine only ensures that the correct mode is running. The routines for modes 0 and 1 share the same initialization and completion sections, with the main body of code being different.

The initialization routine is similar to many HSI routines. The flags are checked to ensure that the HSI FIFO data is valid, and then the FIFO is read. Next, the main body of code (for either mode 0 or mode 1) is

run. At the end time and count values are saved and the holding register is checked for another event. Listing 4-8 contains the initialization and completion sections of the HSI routine.

Listing 4-9 is the main body of the Mode 1 routine. Before any calculations are done in Mode 1, the incoming pulse period is measured to see if it is too fast or too slow for mode 1. The time period between two edges is used so that the duty cycle of the waveform will not affect mode switching. If it is determined that Mode 2 should be set, Port 1.1 is set, all of the HSI lines are disabled, and the HSI fifo is cleared. If Mode 0 is to be set all of the HSI lines are enabled and the variable *LAST_STAT* is cleared. *LAST_STAT* = 0 is used as a flag to indicate the first HSI interrupt in Mode 0 after Mode 1. After the mode checking and setting are complete the incremental value in Timer 2 is used to update

POSITION. The program then returns to the completion section of the routine.

There is a lot more code used in Mode 0 than in Mode 1, most of which is due to the multiple jump statements that determine the current and previous state of the HSI pins. In order to save execution time several blocks of code are repeated as can be seen in Listing 4-10. The first determination is that of which edge had occurred. If a Phase A edge was detected the *LAST1_TIME* and *LAST2_TIME* variables are updated so a reference to the pulse frequency will be available. These are the same variables used under Mode 1. A test is also made to see if the edges are coming fast enough to warrant being in Mode 1, if they are, the switch is made. If the last edge detected was on Phase B, the information is used only to determine direction.

```

In_mode_1:                ; mode 1 HSI routine

    andb    tmpl,hsi_s0,$01010000B
    jne     no_cnt
    cmp_time:                ; Procedure which sets mode 1 also
                                ; sets times to pass the tests

    ld      last2_time,last1_time
    ld      last1_time,time
    cmp1:   sub     tmpl,time,last2_time
    cmp     tmpl,min_hsal
    jh      check_max_time

set_mode_2:
    orb     Port1,$000000010B    ; Set Pl.1 (in mode 2)
    ldb     IOCO,$000000000B    ; Disable all HSI
    mt_hsal: ld      zero,hsal_time
                                ; empty the hsal fifo
    andb    ioal_bak,$01111111B ; clear bit 7
    orb     ioal_bak,ioal
    jbs     ioal_bak,7,mt_hsal ; If hsal is triggered then clear hsal
    br      done_chk

check_max_time:
    sub     tmpl,time,last2_time
    cmp     tmpl,max_hsal
    jnh     done_chk
                                ; max_hsal = addition to min_hsal for
                                ; total time

set_mode_0:
    andb    Port1,$111111000B    ; clear Pl.0,1 set mode 0)
    ldb     IOCO,$010101010B    ; Enable all HSI
    ldb     last_stat,zero

done_chk:
    sub     delta_p,timer_2,tmr2_old ; get timer2 count difference
    jbc     direct,0,add_rev

add_fwd:
    add     position,delta_p
    addc    position+2,zero
    load_last5

add_rev:
    sub     position,delta_p
    subc    position+2,zero
    load_last5

Sselect

```

270061-54

Listing 4-9. Motor Control Mode 1 Routines

270061-55

6-47


```

chk_dir:  cmp     pos_err+2,zero
          jge     go_forward

go_backward: neg     pos_err      ; Pos_err = ABS_VAL (pos_err)
            ldb     pwm_dir,$00h
            cmp     pos_err+2,$0ffffH
            ld     max
            jne     chk_brk
            br

go_forward: ldb     pwm_dir,$01h
            cmp     pos_err+2,zero
            je      chk_brk

ld_max:   ldb     pwm_pwr,max_pwr
          br

chk_brk:  ; Position_Error now = ABS(pos_err)
          cmp     pos_err,pos_pnt
          jnh     hold_position
          cmp     pos_err,brk_pnt
          jh      ld_max

braking:  cmp     pos_delta,zero
          jge     chk_delta
          neg     pos_delta

chk_delta: cmp     pos_delta,vel_pnt
           ; velocity = pos_delta/sample_time
           ; jmp if ABS(velocity) < vel_pnt
           jnh     hold_position

brake:    ldb     pwm_pwr,max_brk
          ldb     tmp,direct
          notb    tmp
          ldb     pwm_dir,tmp

          br      ld_pwr

Hold_position: ; position hold mode
            cmp     pos_err,$02
            calc_out ; if position error < 2 then turn off power
            jh
            clr     tmp+2
            boost
            BR      output

calc_out:  mulub    tmp,max_hold,$255
           mulu     tmp,pos_err      ; Tmp = pos_err * max_hold
           cmp     pos_delta,zero
           jne     no_bst
           add     tmp+2,$04          ; Boost is integral control
           add     tmp+2,boost       ; TMP+2 = MSB(pos_err*max_hold)
           br      ck_max

no_bst:    clr     boost
ck_max:    cmp     tmp+2,max_hold
           jnh     output
           ld      tmp+2,max_hold
maxed:     ld      tmp+2,max_hold
output:    ldb     pwm_pwr,tmp+2

chk_sanity: br      ld_pwr

ld_pwr:    ldb     rpwr,pwm_pwr
           notb    rpwr
           jbs     pwm_dir,0,p2fwd

p2bkwd:    DI
           andb    port2,$01111111B
           ldb     EI
           br      pwrset

p2fwd:     DI
           orb     port2,$10000000B ; set P2.7
           ldb     EI
           pwr_control,rpwr

```

Listing 4-11b: Motor Control Power Algorithm

270061-57

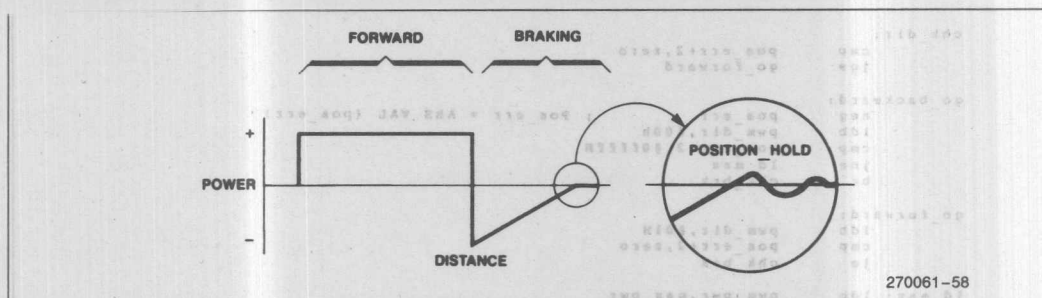


Figure 4-7. Motor Control Modes

error does not get smaller. Once the error does get smaller, usually because the motor starts moving, BOOST is cleared.

A sanity check can be performed at this point to double check that the 8096 has proper control of the motor. In the example the worst that can happen is the proto-

```

pwrset:  cmp     time_err+2,zero    ; do pos_table when err is negative
        jgt     end_p
        br      end_p
        cmp     nxt_pos,#(32+pos_table)
        jlt     get_vals           ; jump if lower
        ld      nxt_pos,pos_table
        clr     time+2
        get_vals:
        ld      des_pos,[nxt_pos]+
        ld      des_pos+2,[nxt_pos]+
        ld      des_time+2,[nxt_pos]+
        ld      max_pwr,[nxt_pos]+
        ld      max_brk,max_pwr
        add     des_pos,offset
        addc    des_pos+2,zero
        sub     last_pos_err,des_pos,position

end_p:   andb    port1,#01111111B    ; clear P1.7
        popf
        ret

pos_table:
        dcl     00000000H           ; position 0
        dcw     0020H, 0080H        ; next time, power
        dcl     0000c000H           ; position 1
        dcw     0040H, 0040H        ; next time, power
        dcl     00000000H           ; position 2
        dcw     0060H, 00c0H        ; next time, power
        dcl     0FFFF8000H          ; position 3
        dcw     0080H, 0080H        ; next time, power

        dcl     00000800H           ; position 4
        dcw     0058H, 0080H        ; next time, power
        dcl     00003000H           ; position 5
        dcw     0070H, 00ffH        ; next time, power
        dcl     00000000H           ; position 6
        dcw     0090H, 00f0H        ; next time, power
        dcl     00000000H           ; position 7
        dcw     0091H, 00f0H        ; next time, power

```

Listing 4-12. Motor Control Next Position Lookup


```

;=====
; SOFTWARE TIMER ROUTINE 2
;=====
CSEG AT 2380H
;=====
; awt2 expired:
; pushf
; ldb hso_command, $3AH
; add hso_time, timer1, swt2_dly
;=====
; orb port1, $00000100B
; cmp out_ptr, $7FFFH
; bnh pulsing
; ld out_ptr, $1F0H
;=====
; pulsing:
; jbc tr_col, 0, swt2_done
;=====
; st position+2, [out_ptr]+
; st position, [out_ptr]+
;=====
; st direct, [out_ptr]+
; st pwm_pwr, [out_ptr]+
;=====
; store 8 bytes externally
;=====
swt2_done:
; sub tmp1, timer1, last1_time
; cmp tmp1, $1800H
; jnh swt2_ret ; keep (time_last4_time) < 7000H
;=====
; add last1_time, $1000H
;=====
swt2_ret:
; andb port1, $11111011B ; clear port1.2
; popf
; ret
;=====

```

Listing 4-15: Motor Control Software Timer 2 Routine

The last routine, shown in Listing 4-15, is the Software Timer 2 routine which outputs some variables to external RAM. It also keeps LAST1_Time within 1800H of Timer1 to prevent overflows from occurring when the Mode 0 and Mode 1 software check this variable.

A complete listing of the program as it is used in our lab can be found in Appendix D. For a given motor or encoder it will probably be necessary to change some of the time constants on the first page of the listing. With the motor used in our experimentation, pulses are missed from time to time when direction changes quickly. If the motor were not as fast to turn around or the encoder were mounted better these problems should disappear. The missing pulses occur when switching from Mode 1 to Mode 0, other than that no anomalies were found in the lab.

Prior to the version of code just discussed, several attempts were made, one of which could be used under certain constraints. It is possible to use only modes 2 and 0 to monitor the encoder, provided the encoder

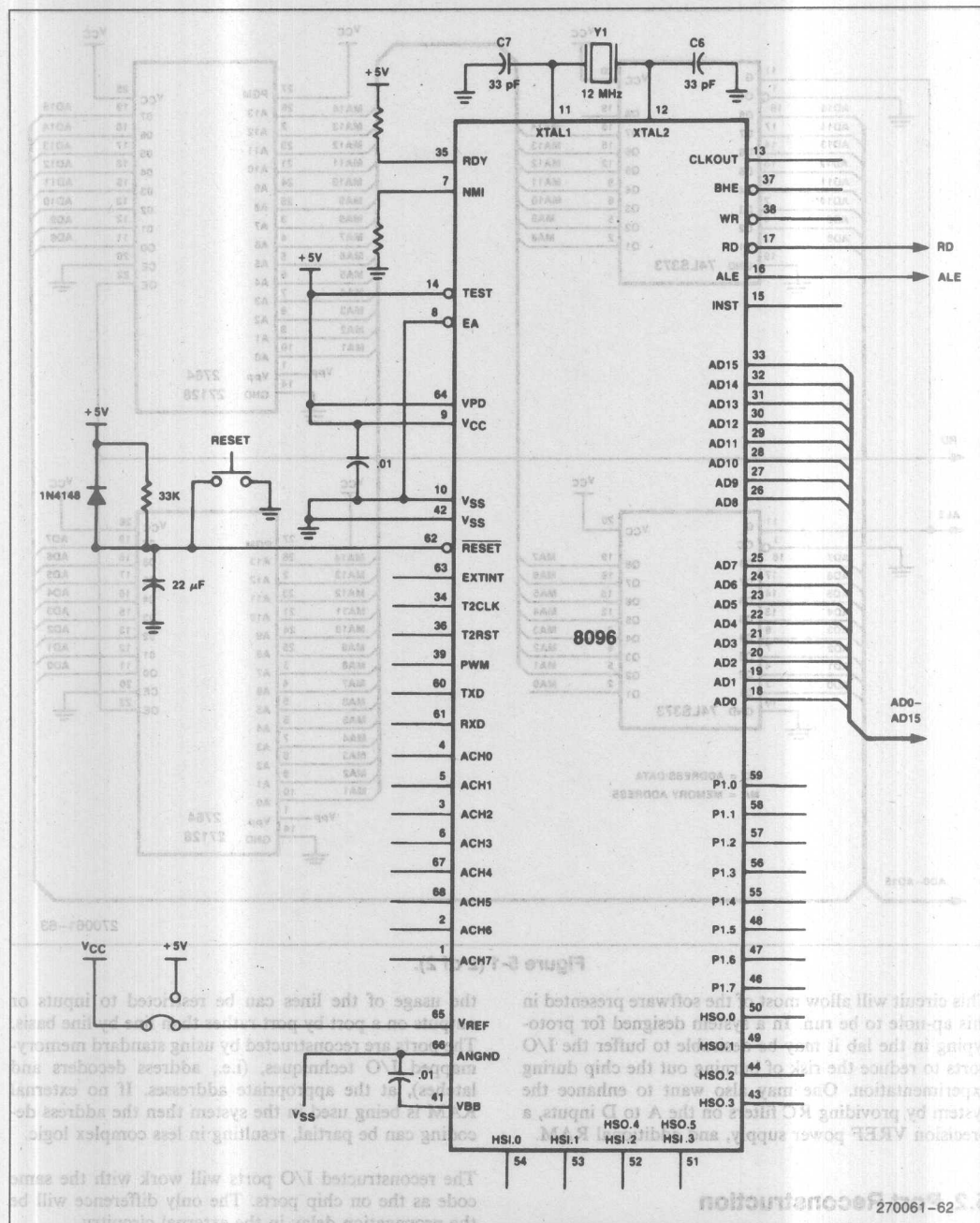
always operates smoothly and provides at least 200 microseconds between the last several edges of Phase A before reversing. This idea was originally tried because the motor was not characterized thoroughly at first, and caused problems because of the motors tendency to stop suddenly when its speed was low.

If an encoder has a lower line count and therefore more time between output pulses the two mode solution can be used. The software for the two mode version can be easily extracted from the three mode version, so it will not be presented.

5.0 HARDWARE EXAMPLE

5.1. EPROM Only Minimum System

The diagram in Figure 5-1 illustrates how to connect an 8096 in a minimum configuration system. Either 2764s or 27128s can be used in the system. Note that the lower EPROM contains the even bytes while the upper



one contains the odd bytes, and the addressing is not fully decoded. This means that the addressing on a 2764 will be such that the lower 4K of each EPROM is mapped at 0000H and 4000H while the upper

4K is mapped at 2000H. If the program being loaded is 16 Kbytes long the first half is loaded into the second half of the 2764s and vice versa. A similar situation exists when using 27128s.

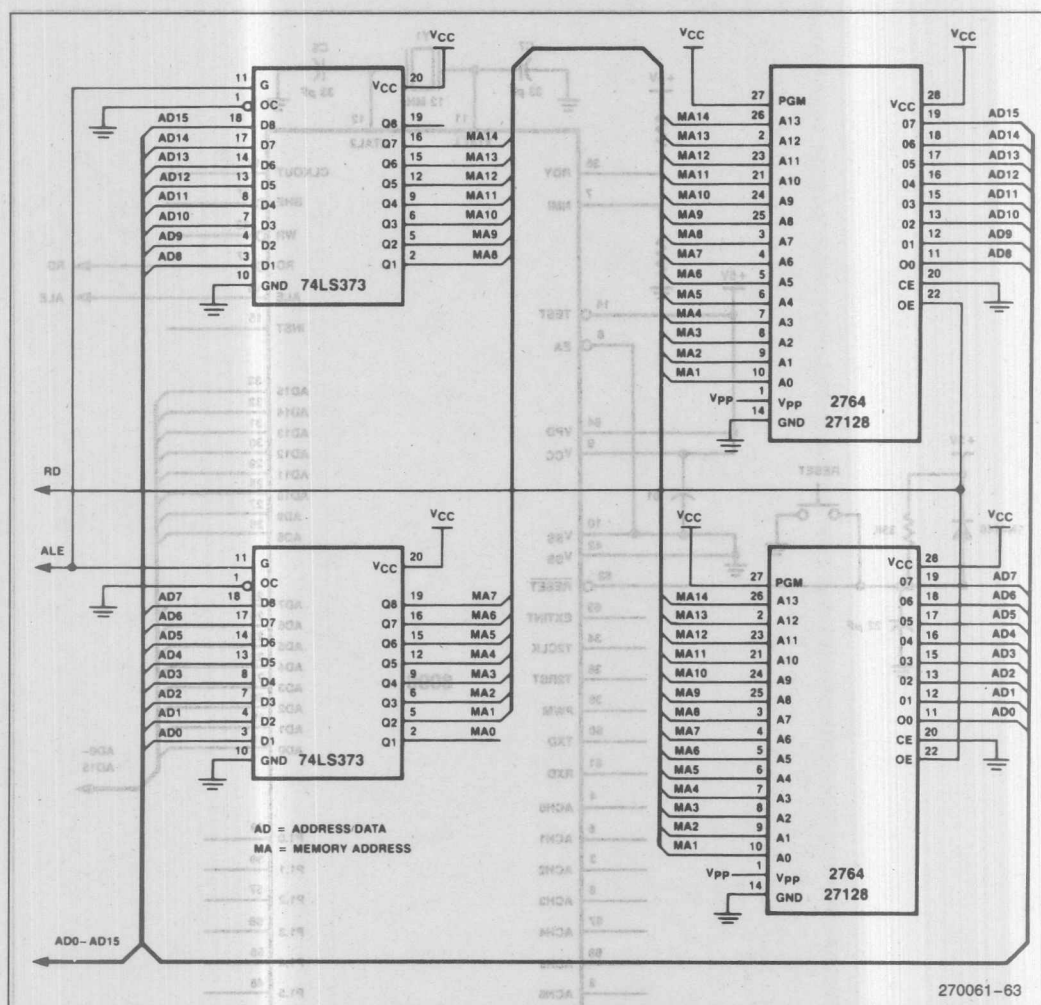


Figure 5-1 (2 of 2).

This circuit will allow most of the software presented in this ap-note to be run. In a system designed for prototyping in the lab it may be desirable to buffer the I/O ports to reduce the risk of burning out the chip during experimentation. One may also want to enhance the system by providing RC filters on the A to D inputs, a precision VREF power supply, and additional RAM.

5.2. Port Reconstruction

If it is desired to fully emulate a 8396 then I/O ports 3 and 4 must be reconstructed. It is easiest to do this if

the usage of the lines can be restricted to inputs or outputs on a port by port rather than line by line basis. The ports are reconstructed by using standard memory-mapped I/O techniques, (i.e., address decoders and latches), at the appropriate addresses. If no external RAM is being used in the system then the address decoding can be partial, resulting in less complex logic.

The reconstructed I/O ports will work with the same code as the on chip ports. The only difference will be the propagation delay in the external circuitry.

VHARDO APPENDIX A BASIC SOFTWARE EXAMPLES

NOISUJWOC 0.3

SERIES-III MCS-96 MACRO ASSEMBLER, V1 0

SOURCE FILE F3:INTER1 A96

OBJECT FILE F3:INTER1 OBJ

CONTROLS SPECIFIED IN INVOCATION COMMAND: NOSB

ERR LOC	OBJECT	LINE	SOURCE STATEMENT
		1	\$TITLE('INTER1 A96: Interpolation routine 1')
		2	;;;;; 8096 Assembly code for table lookup and interpolation
		3	
		4	\$INCLUDE(:FO:DEMO96.INC) ; Include demo definitions
=1		5	\$nolist ; Turn listing off for include file
=1		53	; End of include file
		54	
	0022	55	RSEG at 22H
		56	
	0022	57	IN_VAL: dsb 1 ; Actual Input Value
	0024	58	TABLE_LOW: dsb 1
	0026	59	TABLE_HIGH: dsb 1
	0028	60	IN_DIF: dsb 1 ; Upper Input - Lower Input
	0028	61	IN_DIFB equ IN_DIF : byte
	002A	62	TAB_DIF: dsb 1 ; Upper Output - Lower Output
	002C	63	OUT: dsb 1
	002E	64	RESULT: dsb 1
	0030	65	OUT_DIF: dsl 1 ; Delta Out
		66	
		67	
	2080	68	CSEG at 2080H
		69	
	2080 A1000118	70	LD SP, #100H
		71	
	2084 B0221C	72	look: LDB AL, IN_VAL ; Load temp with Actual Value
	2087 18031C	73	SHRB AL, #3 ; Divide the byte by 8
	208A 71FE1C	74	ANDB AL, #11111110B ; Insure AL is a word address
		75	; This effectively divides AL by 2
		76	; so AL = IN_VAL/16
		77	
	208D AC1C1C	78	LDBZE AX, AL ; Load byte AL to word AX
	2090 A31D002124	79	LD TABLE_LOW, TABLE [AX] ; TABLE_LOW is loaded with the value
		80	; in the table at table location AX
		81	

270061-64

270061-65

SERIES-III MCS-96 MACRO ASSEMBLER. V1.0

SOURCE FILE F3 INTER2 A96

OBJECT FILE F3 INTER2 OBJ

CONTROLS SPECIFIED IN INVOCATION COMMAND NOSB

```

ERR LOC  OBJECT          LINE      SOURCE STATEMENT
1          *TITLE('INTER2 A96  Interpolation routine 2')
2
3          .....      8096 Assembly code for table lookup and interpolation
4          .....      Using tabled values in place of division
5
6          *INCLUDE( F0.DEMO96 INC) , Include demo definitions
7          $nolist      Turn listing off for include file
8          $END          End of include file
9
10         V22EMPTA COMPLETED MD E860K(2)
11
12         0024 0010          155      RSEG at 24H
13         0018 000000+0003+0055 156
14         0024 00180010001+0000 157      IN_VAL: 1800H dsw 1 ; Actual Input Value
15         0026 000000+0001+0018 158      TABLE_LOW: 000H dsw 1 ; Table value for function
16         0028 000000+0003+00+C 159      TABLE_INC: 00H dsw 1 ; Incremental change in function
17         002A          160      IN_DIF: dsw 1 ; Upper Input - Lower Input
18         0002A          161      IN_DIFB equ IN_DIF byte
19         002C          162      OUT: dsw 1
20         002E          163      RESULT: dsw 1
21         0030 3ACB          164      OUT_DIF: dsl 1 ; Delta Out
22
23         0001 005E3C          165      CSEG at 2080H
24         2080          166
25         0001 00005C          167
26         2080 A100011B          168      LD SP, #100H ; Initialize SP to top of reg. file
27
28         2084 B0241C          169      look: LDB AL, IN_VAL ; Load temp with Actual Value
29         2087 18031C          170      SHRB AL, #3 ; Divide the byte by 8
30         208A 71FE1C          171      ANDB AL, #1111110B ; Insure AL is a word address
31         0001 000030          172      ; This effectively divides AL by 2
32         208D AC1C1C          173      LDBZ AX, AL ; Load byte AL to word AX
33         2090 A31D002126          174      LD TABLE_LOW, VAL_TABLE[AX] ; TABLE_LOW is loaded with the value
34         0001 000030          175      ; in the value table at location AX
35         2095 A31D222128          176      LD TABLE_INC, INC_TABLE[AX] ; TABLE_INC is loaded with the value
36         0001 000030          177      ; in the increment table at
37         2098 000030          178      ; location AX+2 HIGH-1000
38         0001 000030          179
39         20A2 000030          180
40         0001 000030          181
41         20A5 000030          182
42         0001 000030          183
43         20A8 000030          184
44         0001 000030          185
45         20AB 000030          186
46         0001 000030          187
47         20AE 000030          188
48         0001 000030          189
49         20B1 000030          190
50         0001 000030          191
51         20B4 000030          192
52         0001 000030          193
53         20B7 000030          194
54         0001 000030          195
55         20BA 000030          196
56         0001 000030          197
57         20BD 000030          198
58         0001 000030          199
59         20C0 000030          200
60         0001 000030          201
61         20C3 000030          202
62         0001 000030          203
63         20C6 000030          204
64         0001 000030          205
65         20C9 000030          206
66         0001 000030          207
67         20CC 000030          208
68         0001 000030          209
69         20CF 000030          210
70         0001 000030          211
71         20D2 000030          212
72         0001 000030          213
73         20D5 000030          214
74         0001 000030          215
75         20D8 000030          216
76         0001 000030          217
77         20DB 000030          218
78         0001 000030          219
79         20DE 000030          220
80         0001 000030          221
81         20E1 000030          222
82         0001 000030          223
83         20E4 000030          224
84         0001 000030          225
85         20E7 000030          226
86         0001 000030          227
87         20EA 000030          228
88         0001 000030          229
89         20ED 000030          230
90         0001 000030          231
91         20F0 000030          232
92         0001 000030          233
93         20F3 000030          234
94         0001 000030          235
95         20F6 000030          236
96         0001 000030          237
97         20F9 000030          238
98         0001 000030          239
99         20FC 000030          240
100        0001 000030          241
101        20FF 000030          242
102        0001 000030          243
103        2102 000030          244
104        0001 000030          245
105        2105 000030          246
106        0001 000030          247
107        2108 000030          248
108        0001 000030          249
109        210B 000030          250
110        0001 000030          251
111        210E 000030          252
112        0001 000030          253
113        2111 000030          254
114        0001 000030          255
115        2114 000030          256
116        0001 000030          257
117        2117 000030          258
118        0001 000030          259
119        211A 000030          260
120        0001 000030          261
121        211D 000030          262
122        0001 000030          263
123        2120 000030          264
124        0001 000030          265
125        2123 000030          266
126        0001 000030          267
127        2126 000030          268
128        0001 000030          269
129        2129 000030          270
130        0001 000030          271
131        212C 000030          272
132        0001 000030          273
133        212F 000030          274
134        0001 000030          275
135        2132 000030          276
136        0001 000030          277
137        2135 000030          278
138        0001 000030          279
139        2138 000030          280
140        0001 000030          281
141        213B 000030          282
142        0001 000030          283
143        213E 000030          284
144        0001 000030          285
145        2141 000030          286
146        0001 000030          287
147        2144 000030          288
148        0001 000030          289
149        2147 000030          290
150        0001 000030          291
151        214A 000030          292
152        0001 000030          293
153        214D 000030          294
154        0001 000030          295
155        2150 000030          296
156        0001 000030          297
157        2153 000030          298
158        0001 000030          299
159        2156 000030          300
160        0001 000030          301
161        2159 000030          302
162        0001 000030          303
163        215C 000030          304
164        0001 000030          305
165        215F 000030          306
166        0001 000030          307
167        2162 000030          308
168        0001 000030          309
169        2165 000030          310
170        0001 000030          311
171        2168 000030          312
172        0001 000030          313
173        216B 000030          314
174        0001 000030          315
175        216E 000030          316
176        0001 000030          317
177        2171 000030          318
178        0001 000030          319
179        2174 000030          320
180        0001 000030          321
181        2177 000030          322
182        0001 000030          323
183        217A 000030          324
184        0001 000030          325
185        217D 000030          326
186        0001 000030          327
187        2180 000030          328
188        0001 000030          329
189        2183 000030          330
190        0001 000030          331
191        2186 000030          332
192        0001 000030          333
193        2189 000030          334
194        0001 000030          335
195        218C 000030          336
196        0001 000030          337
197        218F 000030          338
198        0001 000030          339
199        2192 000030          340
200        0001 000030          341
201        2195 000030          342
202        0001 000030          343
203        2198 000030          344
204        0001 000030          345
205        219B 000030          346
206        0001 000030          347
207        219E 000030          348
208        0001 000030          349
209        21A1 000030          350
210        0001 000030          351
211        21A4 000030          352
212        0001 000030          353
213        21A7 000030          354
214        0001 000030          355
215        21AA 000030          356
216        0001 000030          357
217        21AD 000030          358
218        0001 000030          359
219        21B0 000030          360
220        0001 000030          361
221        21B3 000030          362
222        0001 000030          363
223        21B6 000030          364
224        0001 000030          365
225        21B9 000030          366
226        0001 000030          367
227        21BC 000030          368
228        0001 000030          369
229        21BF 000030          370
230        0001 000030          371
231        21C2 000030          372
232        0001 000030          373
233        21C5 000030          374
234        0001 000030          375
235        21C8 000030          376
236        0001 000030          377
237        21CB 000030          378
238        0001 000030          379
239        21CE 000030          380
240        0001 000030          381
241        21D1 000030          382
242        0001 000030          383
243        21D4 000030          384
244        0001 000030          385
245        21D7 000030          386
246        0001 000030          387
247        21DA 000030          388
248        0001 000030          389
249        21DD 000030          390
250        0001 000030          391
251        21E0 000030          392
252        0001 000030          393
253        21E3 000030          394
254        0001 000030          395
255        21E6 000030          396
256        0001 000030          397
257        21E9 000030          398
258        0001 000030          399
259        21EC 000030          400
260        0001 000030          401
261        21EF 000030          402
262        0001 000030          403
263        21F2 000030          404
264        0001 000030          405
265        21F5 000030          406
266        0001 000030          407
267        21F8 000030          408
268        0001 000030          409
269        21FB 000030          410
270        0001 000030          411
271        21FE 000030          412
272        0001 000030          413
273        2201 000030          414
274        0001 000030          415
275        2204 000030          416
276        0001 000030          417
277        2207 000030          418
278        0001 000030          419
279        220A 000030          420
280        0001 000030          421
281        220D 000030          422
282        0001 000030          423
283        2210 000030          424
284        0001 000030          425
285        2213 000030          426
286        0001 000030          427
287        2216 000030          428
288        0001 000030          429
289        2219 000030          430
290        0001 000030          431
291        221C 000030          432
292        0001 000030          433
293        221F 000030          434
294        0001 000030          435
295        2222 000030          436
296        0001 000030          437
297        2225 000030          438
298        0001 000030          439
299        2228 000030          440
300        0001 000030          441
301        222B 000030          442
302        0001 000030          443
303        222E 000030          444
304        0001 000030          445
305        2231 000030          446
306        0001 000030          447
307        2234 000030          448
308        0001 000030          449
309        2237 000030          450
310        0001 000030          451
311        223A 000030          452
312        0001 000030          453
313        223D 000030          454
314        0001 000030          455
315        2240 000030          456
316        0001 000030          457
317        2243 000030          458
318        0001 000030          459
319        2246 000030          460
320        0001 000030          461
321        2249 000030          462
322        0001 000030          463
323        224C 000030          464
324        0001 000030          465
325        224F 000030          466
326        0001 000030          467
327        2252 000030          468
328        0001 000030          469
329        2255 000030          470
330        0001 000030          471
331        2258 000030          472
332        0001 000030          473
333        225B 000030          474
334        0001 000030          475
335        225E 000030          476
336        0001 000030          477
337        2261 000030          478
338        0001 000030          479
339        2264 000030          480
340        0001 000030          481
341        2267 000030          482
342        0001 000030          483
343        226A 000030          484
344        0001 000030          485
345        226D 000030          486
346        0001 000030          487
347        2270 000030          488
348        0001 000030          489
349        2273 000030          490
350        0001 000030          491
351        2276 000030          492
352        0001 000030          493
353        2279 000030          494
354        0001 000030          495
355        227C 000030          496
356        0001 000030          497
357        227F 000030          498
358        0001 000030          499
359        2282 000030          500
360        0001 000030          501
361        2285 000030          502
362        0001 000030          503
363        2288 000030          504
364        0001 000030          505
365        228B 000030          506
366        0001 000030          507
367        228E 000030          508
368        0001 000030          509
369        2291 000030          510
370        0001 000030          511
371        2294 000030          512
372        0001 000030          513
373        2297 000030          514
374        0001 000030          515
375        229A 000030          516
376        0001 000030          517
377        229D 000030          518
378        0001 000030          519
379        22A0 000030          520
380        0001 000030          521
381        22A3 000030          522
382        0001 000030          523
383        22A6 000030          524
384        0001 000030          525
385        22A9 000030          526
386        0001 000030          527
387        22AC 000030          528
388        0001 000030          529
389        22AF 000030          530
390        0001 000030          531
391        22B2 000030          532
392        0001 000030          533
393        22B5 000030          534
394        0001 000030          535
395        22B8 000030          536
396        0001 000030          537
397        22BB 000030          538
398        0001 000030          539
399        22BE 000030          540
400        0001 000030          541
401        22C1 000030          542
402        0001 000030          543
403        22C4 000030          544
404        0001 000030          545
405        22C7 000030          546
406        0001 000030          547
407        22CA 000030          548
408        0001 000030          549
409        22CD 000030          550
410        0001 000030          551
411        22D0 000030          552
412        0001 000030          553
413        22D3 000030          554
414        0001 000030          555
415        22D6 000030          556
416        0001 000030          557
417        22D9 000030          558
418        0001 000030          559
419        22DC 000030          560
420        0001 000030          561
421        22DF 000030          562
422        0001 000030          563
423        22E2 000030          564
424        0001 000030          565
425        22E5 000030          566
426        0001 000030          567
427        22E8 000030          568
428        0001 000030          569
429        22EB 000030          570
430        0001 000030          571
431        22EE 000030          572
432        0001 000030          573
433        22F1 000030          574
434        0001 000030          575
435        22F4 000030          576
436        0001 000030          577
437        22F7 000030          578
438        0001 000030          579
439        22FA 000030          580
440        0001 000030          581
441        22FD 000030          582
442        0001 000030          583
443        2300 000030          584
444        0001 000030          585
445        2303 000030          586
446        0001 000030          587
447        2306 000030          588
448        0001 000030          589
449        2309 000030          590
450        0001 000030          591
451        230C 000030          592
452        0001 000030          593
453        230F 000030          594
454        0001 000030          595
455        2312 000030          596
456        0001 000030          597
457        2315 000030          598
458        0001 000030          599
459        2318 000030          600
460        0001 000030          601
461        231B 000030          602
462        0001 000030          603
463        231E 000030          604
464        0001 000030          605
465        2321 000030          606
466        0001 000030          607
467        2324 000030          608
468        0001 000030          609
469        2327 000030          610
470        0001 000030          611
471        232A 000030          612
472        0001 000030          613
473        232D 000030          614
474        0001 000030          615
475        2330 000030          616
476        0001 000030          617
477        2333 000030          618
478        0001 000030          619
479        2336 000030          620
480        0001 000030          621
481        2339 000030          622
482        0001 000030          623
483        233C 000030          624
484        0001 000030          625
485        233F 000030          626
486        0001 000030          627
487        2342 000030          628
488        0001 000030          629
489        2345 000030          630
490        0001 000030          631
491        2348 000030          632
492        0001 000030          633
493        234B 000030          634
494        0001 000030          635
495        234E 000030          636
496        0001 000030          637
497        2351 000030          638
498        0001 000030          639
499        2354 000030          640
500        0001 000030          641
501        2357 000030          642
502        0001 000030          643
503        235A 000030          644
504        0001 000030          645
505        235D 000030          646
506        0001 000030          647
507        2360 000030          648
508        0001 000030          649
509        2363 000030          650
510        0001 000030          651
511        2366 000030          652
512        0001 000030          653
513        2369 000030          654
514        0001 000030          655
515        236C 000030          656
516        0001 000030          657
517        236F 000030          658
518        0001 000030          659
519        2372 000030          660
520        0001 000030          661
521        2375 000030          662
522        0001 000030          663
523        2378 000030          664
524        0001 000030          665
525        237B 000030          666
526        0001 000030          667
527        237E 000030          668
528        0001 000030          669
529        2381 000030          670
530        0001 000030          671
531        2384 000030          672
532        0001 000030          673
533        2387 000030          674
534        0001 000030          675
535        238A 000030          676
536        0001 000030          677
537        238D 000030          678
538        0001 000030          679
539        2390 000030          680
540        0001 000030          681
541        2393 000030          682
542        0001 000030          683
543        2396 000030          684
544        0001 000030          685
545        2399 000030          686
546        0001 000030          687
547        239C 000030          688
548        0001 000030          689
549        239F 000030          690
550        0
```

```

209A 510F242A      87      ANDB    IN_DIFB, IN_VAL, #0FH ; IN_DIFB=least significant 4 bits
                                     ; of IN_VAL
209E AC2A2A        89      LDBZ    IN_DIF, IN_DIFB ; Load byte IN_DIFB to word IN_DIF
20A1 FE4C282A30    91      MUL     OUT_DIF, IN_DIF, TABLE_INC
                                     ; Output_difference =
                                     ; Input_difference*Incremental_change
20A6 4426302C      93      ADD     OUT, OUT_DIF, TABLE_LOW ; Add output difference to output
                                     ; generated with truncated IN_VAL
                                     ; as input
20AA 08042C        98      SHR     OUT, #4 ; Round to 12-bit answer
20AD A4002C        99      ADDC    OUT, zero ; Round up if Carry = 1
20B0 C02E2C        101     no_inc: ST     OUT, RESULT ; Store OUT to RESULT
20B3 27CF          102     BR      look ; Branch to "look:"
2100              105     cseg    AT 2100H
2100              107     val_table:
2100 000000200034004C 108     DCW     0000H, 2000H, 3400H, 4C00H ; A random function
2108 005D006A0072007B 109     DCW     5D00H, 6A00H, 7200H, 7B00H
2110 007B007D0076006D 110     DCW     7B00H, 7D00H, 7600H, 6D00H
2118 005D004800340022 111     DCW     5D00H, 4800H, 3400H, 2200H
2120 0010          112     DCW     1000H
2122              113     inc_table:
2122 0002400180011001 114     DCW     0200H, 0140H, 0180H, 0110H ; Table of incremental
212A D000800060003000 115     DCW     00D0H, 0080H, 0060H, 0030H ; differences
2132 200090FF70FF00FF 116     DCW     00020H, 0FF90H, 0FF70H, 0FF00H
213A E0FE90FEE0FEE0FE 117     DCW     0FEE0H, 0FE90H, 0FEE0H, 0FEE0H
2142              118     END
2142              119
ASSEMBLY COMPLETED, NO ERROR(S) FOUND.

```

270061-67

SERIES-III PL/M-96 V1 0 COMPILATION OF MODULE PLMEX
 OBJECT MODULE PLACED IN F3 PLMEX1 OBJ
 COMPILER INVOKED BY: PLM96 86 F3 PLMEX1 P96 CODE

\$TITLE('PLMEX1: PLM-96 Example Code for Table Lookup')

/* PLM-96 CODE FOR TABLE LOOK-UP AND INTERPOLATION */

```

1      PLMEX:      DO;

2      1      DECLARE IN_VAL      WORD      PUBLIC;
3      1      DECLARE TABLE_LOW  INTEGER   PUBLIC;
4      1      DECLARE TABLE_HIGH INTEGER   PUBLIC;
5      1      DECLARE TABLE_DIF  INTEGER   PUBLIC;
6      1      DECLARE OUT         INTEGER   PUBLIC;
7      1      DECLARE RESULT      INTEGER   PUBLIC;
8      1      DECLARE OUT_DIF     LONGINT   PUBLIC;
9      1      DECLARE TEMP        WORD      PUBLIC;

10     1      DECLARE TABLE(17)  INTEGER DATA ( /* A random function */
        0000H, 2000H, 3400H, 4C00H,
        5D00H, 6A00H, 7200H, 7800H,
        7B00H, 7D00H, 7600H, 6D00H,
        5D00H, 4B00H, 3400H, 2200H,
        1000H);

11     1      DMPY: PROCEDURE (A,B) LONGINT EXTERNAL;
12     2      DECLARE (A,B) INTEGER;
13     2      END DMPY;

14     1      LOOP
        TEMP=SHR(IN_VAL,4); /* TEMP is the most significant 4 bits of IN_VAL */

15     1      TABLE_LOW=TABLE(TEMP); /* If "TEMP" was replaced by "SHR(IN_VAL,4)" */
16     1      TABLE_HIGH=TABLE(TEMP+1); /* The code would work but the 8096 would */
        /* do two shifts */

17     1      TABLE_DIF=TABLE_HIGH-TABLE_LOW;

18     1      OUT_DIF=DMPY(TABLE_DIF,SIGNED(IN_VAL AND 0FH)) /16;

19     1      OUT=SAR((TABLE_LOW+OUT_DIF),4); /* SAR performs an arithmetic right shift,
        in this case 4 places are shifted */

```

270061-68

270061-69

270061-70

270061-71

SERIES-III MCS-96 MACRO ASSEMBLER, V1.0

OBJECT FILE: :F3:MULT.OBJ

CONTROLS SPECIFIED IN INVOCATION COMMAND: NOSB

ASSEMBLY COMPLETED, NO ERROR(S) FOUND.

270061-72

SERIES-III MCS-96 RELOCATOR AND LINKER, V2.0
Copyright 1983 Intel Corporation

INPUT FILES: :F3:PLMEX1.OBJ, :F3:MULT.OBJ, PLM96.LIB
OUTPUT FILE: :F3:PLMOUT.OBJ
CONTROLS SPECIFIED IN INVOCATION COMMAND:
ROM(2080H-3FFFH)

INPUT MODULES INCLUDED:
:F3:PLMEX1.OBJ(PLMEX) 12/25/84
:F3:MULT.OBJ(MULT) 12/25/84
PLM96.LIB(PLMREG) 11/02/83

SEGMENT MAP FOR :F3:PLMOUT.OBJ(PLMEX):

TYPE	BASE	LENGTH	ALIGNMENT	MODULE NAME
**RESERVED*	0000H	001AH		
*** GAP ***	001AH	0002H		
REG	001CH	0008H	ABSOLUTE	PLMREG
REG	0024H	0012H	WORD	PLMEX
STACK	0036H	0006H	WORD	
*** GAP ***	003CH	2044H		
CODE	2080H	0003H	ABSOLUTE	PLMEX
*** GAP ***	2083H	0001H		
CODE	2084H	007CH	WORD	PLMEX
CODE	2100H	000AH	BYTE	MULT
*** GAP ***	210AH	DEF6H		

270061-15

270061-73

SERIES-III MCS-96 MACRO ASSEMBLER, V1 0

SOURCE FILE F3.PULSE A96

OBJECT FILE F3.PULSE.OBJ

CONTROLS SPECIFIED IN INVOCATION COMMAND NOSB

```

ERR LOC  OBJECT          LINE      SOURCE STATEMENT
      1  $TITLE('PULSE A96: Measuring pulses using the HSI unit')
      2
      3  $INCLUDE(DEMO96 INC)
=1     4  $nolist ; Turn listing off for include file
=1     52 ; End of include file
      53
002B    54  rseg      at 28H
      55
002B    56          HIGH_TIME:      dsw      1
002A    57          LOW_TIME:       dsw      1
002C    58          PERIOD:         dsw      1
002E    59          HI_EDGE:        dsw      1
0030    60          LO_EDGE:        dsw      1
      61
      62
      63
20B0    64  cseg      at 20B0H
      65
      66
20B0 A100011B    67          LD          SP, #100H
20B4 B10115    68  LD      IOCO, #00000001B ; Enable HSI 0
20B7 B10F03    69  LD      HSI_MODE, #00001111B ; HSI 0 look for either edge
      70
20B8 442A282C    71  wait: ADD     PERIOD, HIGH_TIME, LOW_TIME
20BE 3E1603    72  JBS     IOS1, 6, contin ; If FIFO is full
2091 3716F6    73  JBC     IOS1, 7, wait ; Wait while no pulse is entered
      74
2094 B0061C    75  LD      AL, HSI_STATUS ; Load status; Note that reading
      76 ; HSI_TIME clears HSI_STATUS
      77
2097 A00420    78  LD      BX, HSI_TIME ; Load the HSI_TIME
      79
209A 391C09    80  JBS     AL, 1, hsi_hi ; Jump if HSI 0 is high
      81
209D C03020    82  ST      hsi_lo: ST    BX, LO_EDGE
20A0 482E302B    83  SUB     HIGH_TIME, LO_EDGE, HI_EDGE
20A4 27E4      84  BR      wait
      85
20A6 C02E20    86  ST      hsi_hi: ST    BX, HI_EDGE
      87

```

270061-75

SERIES III MCS-96 MACRO ASSEMBLER, V1.0

SOURCE FILE F3 ENHSI A96

OBJECT FILE F3 ENHSI OBJ

CONTROLS SPECIFIED IN INVOCATION COMMAND NOSB

ERR LOC	OBJECT	LINE	SOURCE STATEMENT
		1	\$TITLE ('ENHSI A96 ENHANCED HSI PULSE ROUTINE')
		2	
		3	\$INCLUDE(Demo96 INC)
		4	\$nolist , Turn listing off for include file
		52	End of include file
		53	
0028		54	RSEG AT 28H
		55	
0028		56	TIME DSW 1
002A		57	LAST_RISE DSW 1
002C		58	LAST_FALL DSW 1
002E		59	HSI_SO DSB 1
002F		60	IOS1_BAK DSB 1
0030		61	PERIOD DSW 1
0032		62	LOW_TIME DSW 1
0034		63	HIGH_TIME DSW 1
0036		64	COUNT DSW 1
		65	
2080		66	cseg at 2080H
		67	
2080 A100011B		68	init: LD SP, #100H
		69	
2084 B12516		70	LDB IOC1, #00100101B ; Disable HSO 4, HSO 5, HSI_INT=first,
		71	; Enable PWM, TXD, TIMER1_OVRFLOW_INT
		72	
2087 B19903		73	LDB HSI_MODE, #10011001B ; set hsi.1 -, hsi.0 +
208A B10715		74	LDB IOC0, #00000111B ; Enable hsi 0.1
		75	; T2 CLOCK=T2CLK, T2RST=T2RST
		76	; Clear timer2
		77	
		78	
208D 717F2F		79	wait: ANDB IOS1_BAK, #01111111B ; Clear IOS1_BAK.7
2090 90162F		80	ORB IOS1_BAK, IOS1 ; Store into temp to avoid clearing
		81	; other flags which may be needed
2093 372FF7		82	JBC IOS1_BAK.7, wait ; If hsi is not triggered then
		83	; jump to wait
		84	
2096 5155062E		85	ANDB HSI_SO, HSI_STATUS, #01010101B
209A A0042B 1ED		86	LD TIME, HSI_TIME
		87	
		88	
		89	
		90	
		91	
		92	
		93	
		94	
		95	
		96	
		97	
		98	
		99	
		100	

270061-77

6-69

ASSEMBLY COMPLETED, NO ERROR(S) FOUND

270061-78

SERIES-III MCS-96 MACRO ASSEMBLER, V1.0

SOURCE FILE: F3:HSODRV.A96

OBJECT FILE: F3:HSODRV.OBJ

CONTROLS SPECIFIED IN INVOCATION COMMAND: NOSB

ERR	LOC	OBJECT	LINE	SOURCE STATEMENT
			1	\$TITLE('HSODRV.A96: Driver module for HSO PWM program')
			2	
			3	HSODRV MODULE MAIN, STACKSIZE(8)
			4	
			5	
			6	PUBLIC HSO_ON_0, HSO_OFF_0
			7	PUBLIC HSO_ON_1, HSO_OFF_1
			8	PUBLIC HSO_TIME, HSO_COMMAND
			9	PUBLIC SP, TIMER1, IOSO
			10	
			11	\$INCLUDE(DEMO96.INC)
			12	\$nolist ; Turn listing off for include file
			60	; End of include file
			61	
	0028		62	rseg at 28H
			63	
			64	EXTRN OLD_STAT :byte
			65	
	0028		66	HSO_ON_0: dsw 1
	002A		67	HSO_OFF_0: dsw 1
	002C		68	HSO_ON_1: dsw 1
	002E		69	HSO_OFF_1: dsw 1
	0030		70	count: dsb 1
			71	
	2080		72	cseg at 2080H
			73	
	3001		74	EXTRN wait :entry
			75	
	2080 FA		76	strt: DI
	2081 A100011B		77	LD SP, #100H
	2085 510F1500	E	78	ANDB OLD_STAT, IOSO, #0FH
	2089 950F00	E	79	XORB OLD_STAT, #0FH
	208C		80	
	208C A1000122		81	initial: LD CX, #0100H
			82	
	2090 A100101C		83	
	2094 4B221C20		84	loop: LD AX, #1000H
	2098 A0221C		85	SUB BX, AX, CX
			86	LD AX, CX
			87	

270061-79

270061-80

SERIES-III MCS-96 MACRO ASSEMBLER: V1 0

SOURCE FILE: F3.HSOMOD.A96

OBJECT FILE: F3.HSOMOD.OBJ

CONTROLS SPECIFIED IN INVOCATION COMMAND: NOSB

ERR	LOC	OBJECT	LINE	SOURCE STATEMENT
			1	\$TITLE('HSOMOD.A96: 8096 PWM PROGRAM MODIFIED FOR DRIVER')
			2	\$PAGEWIDTH(130)
			3	
			4	; This program will provide 3 PWM outputs on HSO pins 0-2
			5	; The input parameters passed to the program are:
			6	
			7	HSO_ON_N HSO on time for pin N
			8	HSO_OFF_N HSO off time for pin N
			9	
			10	Where: Times are in timer1 cycles
			11	N takes values from 0 to 3
			12	
			13
			14	
			15	
			16	NOTE: Use this file to replace the declaration section of
			17	the HSO PWM program from "\$INCLUDE(DEMO96.INC)" through
			18	the line prior to the label "wait". Also change the last
			19	branch in the program to a "RET".
			20	
	0000		21	RSEG
			22	
	0000		23	D_STAT: DSB 1
			24	extrn HSO_ON_0 :word , HSO_OFF_0 :word
			25	extrn HSO_ON_1 :word , HSO_OFF_1 :word
			26	extrn HSO_TIME :word , HSO_COMMAND :byte
			27	extrn TIMER1 :word , IOS0 :byte
			28	extrn SP :word
			29	
			30	public OLD_STAT
	0001		31	OLD_STAT: dsb 1
	0002		32	NEW_STAT: dsb 1
			33	
			34	
			35	cseg
			36	PUBLIC wait
			37	
	0000	3E00FD	38	wait: JBS IOS0, 6, wait ; Loop until HSO holding register
	0003	FD	39	NOP ; is empty
			40	
			41	; For operation with interrupts 'store_stat:' would be the
			42	entry point of the routine.
			43	Note that a DI or PUSHF might have to be added.
			44	

270061-81

A.6. PWM Using the HSO (Continued)

6-73

```

0004                                45  store_stat:
0004 510F0002      E  46      ANDB NEW_STAT, IOS0, #0FH      ; Store new status of HSO
0008 980201        R  47      CMPB OLD_STAT, NEW_STAT
0008 DFF3          48      JE wait
000D 940201        R  49      XORB OLD_STAT, NEW_STAT
                                50
                                51
0010                                52  check_0:
0010 300113      R  53      JBC OLD_STAT, 0, check_1      ; Jump if OLD_STAT(0)=NEW_STAT(0)
0013 380209      R  54      JBS NEW_STAT, 0, set_off_0
                                55
0016                                56  set_on_0:
0016 B13000      E  57      LDB HSO_COMMAND, #00110000B      ; Set HSO for timer1, set pin 0
0019 44000000     E  58      ADD HSO_TIME, TIMER1, HSO_OFF_0      ; Time to set pin = Timer1 value
001D 2007          59      BR check_1      ; + Time for pin to be low
                                60
001F                                61  set_off_0:
001F B11000      E  62      LDB HSO_COMMAND, #00010000B      ; Set HSO for timer1, clear pin 0
0022 44000000     E  63      ADD HSO_TIME, TIMER1, HSO_ON_0      ; Time to clear pin = Timer1 value
                                64      ; + Time for pin to be high
0026                                65  check_1:
0026 310113      R  66      JBC OLD_STAT, 1, check_done      ; Jump if OLD_STAT(1)=NEW_STAT(1)
0029 390209      R  67      JBS NEW_STAT, 1, set_off_1
                                68
002C                                69  set_on_1:
002C B13100      E  70      LDB HSO_COMMAND, #00110001B      ; Set HSO for timer1, set pin 1
002F 44000000     E  71      ADD HSO_TIME, TIMER1, HSO_OFF_1      ; Time to set pin = Timer1 value
0033 2007          72      BR check_done
                                73
0035                                74  set_off_1:
0035 B11100      E  75      LDB HSO_COMMAND, #00010001B      ; Set HSO for timer1, clear pin 1
0038 44000000     E  76      ADD HSO_TIME, TIMER1, HSO_ON_1      ; Time to clear pin = Timer1 value
                                77      ; + Time for pin to be high
003C                                78  check_done:
003C B00201      R  79      LDB OLD_STAT, NEW_STAT      ; Store current status and
0038                                80      ; wait for interrupt flag
                                81
003F F0          82      RET
                                83      use "BR wait" if this routine is used with the driver
                                84
0040                                85  END

```

ASSEMBLY COMPLETED. NO ERROR(S) FOUND.

270061-82

SERIES-III MCS-96 MACRO ASSEMBLER, V1 0

SOURCE FILE F3 SP A96

OBJECT FILE F3 SP OBJ

CONTROLS SPECIFIED IN INVOCATION COMMAND NOSB

ERR LOC	OBJECT	LINE	SOURCE STATEMENT
		1	
		2	\$TITLE('SP A96: SERIAL PORT DEMO PROGRAM')
		3	
		4	\$INCLUDE(DEMO96 INC)
		5	\$nolist ; Turn listing off for include file
		53	; End of include file
		54	
	0028	55	rseg at 28H
		56	
	0028	57	CHR: dsb 1
	0029	58	SPTMP: dsb 1
	002A	59	TEMPO: dsb 1
	002B	60	TEMP1: dsb 1
	002C	61	RCV_FLAG: dsb 1
	0030	62	
	200C	63	cseg at 200CH
		64	
	200C	65	DCW set_port_int
		66	
	2080	67	cseg at 2080H
		68	
	2080	69	LD SP, #100H
		70	
	2084	71	LDB IOC1, #00100000H ; Set P2.0 to TXD
		72	
		73	; Baud rate = input frequency / (64*baud_val)
		74	; baud_val = (input frequency/64) / baud rate
		75	
		76	
	0027	77	baud_val equ 39 ; 39 = (12,000,000/64)/4800 baud
		78	
	0080	79	BAUD_HIGH equ ((baud_val-1)/256) OR 80H ; Set MSB to 1
	0026	80	BAUD_LOW equ (baud_val-1) MOD 256
		81	
		82	
	2087	83	LDB BAUD_REG, #BAUD_LOW
	208A	84	LDB BAUD_REG, #BAUD_HIGH
		85	
		86	
		87	
		88	
		89	
		90	
		91	
		92	

270061-83

A.7. Serial Port (Continued)

6-75

```

208D B14911      86      LDB      SPCON, #01001001B      ; Enable receiver, Mode 1
                87
                88
                89
                90
                91
                92      STB      SBUF, CHR      ; Clear serial Port
2090 C42807      93      LDB      TEMPO, #00100000B      ; Set TI-temp
2093 B1202A      94
                95
                96      LDB      INT_MASK, #01000000B      ; Enable Serial Port Interrupt
2096 B14008      97
2099 FB          98      EI
209A 27FE        99      BR      loop      ; Wait for serial port interrupt
                100
                101      ser_port_int
209C F2          102      PUSHF
209D            103      rd_again:
                104      LDB      SPTEMP, SPSTAT      ; This section of code can be replaced
                105      ORB      TEMPO, SPTEMP      ; with "ORB TEMPO, SP_STAT" when the
                106      ANDB      SPTEMP, #01100000B      ; serial port TI and RI bugs are fixed
                107      JNE      rd_again      ; Repeat until TI and RI are properly cleared
20A3 716029      108
20A6 D7F5        109      get_byte:
                110      JBC      TEMPO, 6, put_byte      ; If RI-temp is not set
                111      STB      SBUF, CHR      ; Store byte
                112      ANDB      TEMPO, #10111111B      ; CLR RI-temp
20AB 362A09      113      LDB      RCV_FLAG, #0FFH      ; Set bit-received flag
20AB C42807      114
20AE 71BF2A      115      put_byte:
20B1 B1FF2C      116      JBC      RCV_FLAG, 0, continue      ; If receive flag is cleared
                117      JBC      TEMPO, 5, continue      ; If TI was not set
                118      LDB      SBUF, CHR      ; Send byte
                119      ANDB      TEMPO, #11011111B      ; CLR TI-temp
20B4 302C18      120
20B7 352A15      121      ANDB      CHR, #01111111B      ; This section of code appends
20BA 802807      122      CMPB      CHR, #0DH      ; an LF after a CR is sent
20BD 71DF2A      123      JNE      clr_rcv      ;
                124      LDB      CHR, #0AH      ;
                125      BR      continue
20C0 717F2B      126
20C3 990D2B      127      clr_rcv:
20C6 D705        128      CLRB      RCV_FLAG      ; Clear bit-received flag
20CB B10A2B      129
20CB 2002        130      continue:
20CD            131      POPF
20CD 112C        132      RET
20CF F3          133
20CF F0          134
20D1            135      END

```

ASSEMBLY COMPLETED. NO ERROR(S) FOUND.

270061-84

SERIES-III MCS-96 MACRO ASSEMBLER, V1 0

510091-94

SOURCE FILE: F3:ATDD.A96

OBJECT FILE: F3:ATDD.OBJ

CONTROLS SPECIFIED IN INVOCATION COMMAND: NOSB

```

ERR LOC OBJECT LINE SOURCE STATEMENT
5000 5000 1 $TITLE('ATDD.A96: SCANNING THE A TO D CHANNELS')
5001 5001 2
5002 5002 3
5003 5003 4 $INCLUDE(DEMO96.INC)
5004 5004 5 $nolist ; Turn listing off for include file
5005 5005 6 $end ; End of include file
5006 5006 7
5007 5007 8 RSEG at 2BH
5008 5008 9
5009 5009 10 EQU EQU BX:BYTE
5010 5010 11 EQU EQU DX:BYTE
5011 5011 12
5012 5012 13 RESULT_TABLE:
5013 5013 14 RESULT_1: dsw 1
5014 5014 15 RESULT_2: dsw 1
5015 5015 16 RESULT_3: dsw 1
5016 5016 17 RESULT_4: dsw 1
5017 5017 18
5018 5018 19 cseg at 2080H
5019 5019 20
5020 5020 21 start: LD SP, #100H ; Set Stack Pointer
5021 5021 22 CLR BX
5022 5022 23
5023 5023 24 next: ADDB AD_COMMAND, BL, #1000B ; Start conversion on channel
5024 5024 25 ; indicated by BL register
5025 5025 26
5026 5026 27 NOP ; Wait for conversion to start
5027 5027 28 NOP
5028 5028 29 check: JBS AD_RESULT_LO, 3, check ; Wait while A to D is busy
5029 5029 30
5030 5030 31 LDB AL, AD_RESULT_LO ; Load low order result
5031 5031 32 LDB AH, AD_RESULT_HI ; Load high order result
5032 5032 33
5033 5033 34 ADDB DL, BL, BL ; DL=BL*2
5034 5034 35 LDBZ DX, DL
5035 5035 36 ST AX, RESULT_TABLE[DX] ; Store result indexed by BL*2
5036 5036 37
5037 5037 38 INCB BL ; Increment BL modulo 4
5038 5038 39
5039 5039 40
5040 5040 41
5041 5041 42
5042 5042 43
5043 5043 44
5044 5044 45
5045 5045 46
5046 5046 47
5047 5047 48
5048 5048 49
5049 5049 50
5050 5050 51
5051 5051 52
5052 5052 53
5053 5053 54
5054 5054 55
5055 5055 56
5056 5056 57
5057 5057 58
5058 5058 59
5059 5059 60
5060 5060 61
5061 5061 62
5062 5062 63
5063 5063 64
5064 5064 65
5065 5065 66
5066 5066 67
5067 5067 68
5068 5068 69
5069 5069 70
5070 5070 71
5071 5071 72
5072 5072 73
5073 5073 74
5074 5074 75
5075 5075 76
5076 5076 77
5077 5077 78
5078 5078 79
5079 5079 80
5080 5080 81
5081 5081 82
5082 5082 83
5083 5083 84
5084 5084 85
5085 5085 86
5086 5086 87
5087 5087 88
5088 5088 89
5089 5089 90
5090 5090 91
5091 5091 92
5092 5092 93
5093 5093 94
5094 5094 95
5095 5095 96
5096 5096 97
5097 5097 98
5098 5098 99
5099 5099 100
5100 5100 101
5101 5101 102
5102 5102 103
5103 5103 104
5104 5104 105
5105 5105 106
5106 5106 107
5107 5107 108
5108 5108 109
5109 5109 110
5110 5110 111
5111 5111 112
5112 5112 113
5113 5113 114
5114 5114 115
5115 5115 116
5116 5116 117
5117 5117 118
5118 5118 119
5119 5119 120
5120 5120 121
5121 5121 122
5122 5122 123
5123 5123 124
5124 5124 125
5125 5125 126
5126 5126 127
5127 5127 128
5128 5128 129
5129 5129 130
5130 5130 131
5131 5131 132
5132 5132 133
5133 5133 134
5134 5134 135
5135 5135 136
5136 5136 137
5137 5137 138
5138 5138 139
5139 5139 140
5140 5140 141
5141 5141 142
5142 5142 143
5143 5143 144
5144 5144 145
5145 5145 146
5146 5146 147
5147 5147 148
5148 5148 149
5149 5149 150
5150 5150 151
5151 5151 152
5152 5152 153
5153 5153 154
5154 5154 155
5155 5155 156
5156 5156 157
5157 5157 158
5158 5158 159
5159 5159 160
5160 5160 161
5161 5161 162
5162 5162 163
5163 5163 164
5164 5164 165
5165 5165 166
5166 5166 167
5167 5167 168
5168 5168 169
5169 5169 170
5170 5170 171
5171 5171 172
5172 5172 173
5173 5173 174
5174 5174 175
5175 5175 176
5176 5176 177
5177 5177 178
5178 5178 179
5179 5179 180
5180 5180 181
5181 5181 182
5182 5182 183
5183 5183 184
5184 5184 185
5185 5185 186
5186 5186 187
5187 5187 188
5188 5188 189
5189 5189 190
5190 5190 191
5191 5191 192
5192 5192 193
5193 5193 194
5194 5194 195
5195 5195 196
5196 5196 197
5197 5197 198
5198 5198 199
5199 5199 200
5200 5200 201
5201 5201 202
5202 5202 203
5203 5203 204
5204 5204 205
5205 5205 206
5206 5206 207
5207 5207 208
5208 5208 209
5209 5209 210
5210 5210 211
5211 5211 212
5212 5212 213
5213 5213 214
5214 5214 215
5215 5215 216
5216 5216 217
5217 5217 218
5218 5218 219
5219 5219 220
5220 5220 221
5221 5221 222
5222 5222 223
5223 5223 224
5224 5224 225
5225 5225 226
5226 5226 227
5227 5227 228
5228 5228 229
5229 5229 230
5230 5230 231
5231 5231 232
5232 5232 233
5233 5233 234
5234 5234 235
5235 5235 236
5236 5236 237
5237 5237 238
5238 5238 239
5239 5239 240
5240 5240 241
5241 5241 242
5242 5242 243
5243 5243 244
5244 5244 245
5245 5245 246
5246 5246 247
5247 5247 248
5248 5248 249
5249 5249 250
5250 5250 251
5251 5251 252
5252 5252 253
5253 5253 254
5254 5254 255
5255 5255 256
5256 5256 257
5257 5257 258
5258 5258 259
5259 5259 260
5260 5260 261
5261 5261 262
5262 5262 263
5263 5263 264
5264 5264 265
5265 5265 266
5266 5266 267
5267 5267 268
5268 5268 269
5269 5269 270
5270 5270 271
5271 5271 272
5272 5272 273
5273 5273 274
5274 5274 275
5275 5275 276
5276 5276 277
5277 5277 278
5278 5278 279
5279 5279 280
5280 5280 281
5281 5281 282
5282 5282 283
5283 5283 284
5284 5284 285
5285 5285 286
5286 5286 287
5287 5287 288
5288 5288 289
5289 5289 290
5290 5290 291
5291 5291 292
5292 5292 293
5293 5293 294
5294 5294 295
5295 5295 296
5296 5296 297
5297 5297 298
5298 5298 299
5299 5299 300
5300 5300 301
5301 5301 302
5302 5302 303
5303 5303 304
5304 5304 305
5305 5305 306
5306 5306 307
5307 5307 308
5308 5308 309
5309 5309 310
5310 5310 311
5311 5311 312
5312 5312 313
5313 5313 314
5314 5314 315
5315 5315 316
5316 5316 317
5317 5317 318
5318 5318 319
5319 5319 320
5320 5320 321
5321 5321 322
5322 5322 323
5323 5323 324
5324 5324 325
5325 5325 326
5326 5326 327
5327 5327 328
5328 5328 329
5329 5329 330
5330 5330 331
5331 5331 332
5332 5332 333
5333 5333 334
5334 5334 335
5335 5335 336
5336 5336 337
5337 5337 338
5338 5338 339
5339 5339 340
5340 5340 341
5341 5341 342
5342 5342 343
5343 5343 344
5344 5344 345
5345 5345 346
5346 5346 347
5347 5347 348
5348 5348 349
5349 5349 350
5350 5350 351
5351 5351 352
5352 5352 353
5353 5353 354
5354 5354 355
5355 5355 356
5356 5356 357
5357 5357 358
5358 5358 359
5359 5359 360
5360 5360 361
5361 5361 362
5362 5362 363
5363 5363 364
5364 5364 365
5365 5365 366
5366 5366 367
5367 5367 368
5368 5368 369
5369 5369 370
5370 5370 371
5371 5371 372
5372 5372 373
5373 5373 374
5374 5374 375
5375 5375 376
5376 5376 377
5377 5377 378
5378 5378 379
5379 5379 380
5380 5380 381
5381 5381 382
5382 5382 383
5383 5383 384
5384 5384 385
5385 5385 386
5386 5386 387
5387 5387 388
5388 5388 389
5389 5389 390
5390 5390 391
5391 5391 392
5392 5392 393
5393 5393 394
5394 5394 395
5395 5395 396
5396 5396 397
5397 5397 398
5398 5398 399
5399 5399 400
5400 5400 401
5401 5401 402
5402 5402 403
5403 5403 404
5404 5404 405
5405 5405 406
5406 5406 407
5407 5407 408
5408 5408 409
5409 5409 410
5410 5410 411
5411 5411 412
5412 5412 413
5413 5413 414
5414 5414 415
5415 5415 416
5416 5416 417
5417 5417 418
5418 5418 419
5419 5419 420
5420 5420 421
5421 5421 422
5422 5422 423
5423 5423 424
5424 5424 425
5425 5425 426
5426 5426 427
5427 5427 428
5428 5428 429
5429 5429 430
5430 5430 431
5431 5431 432
5432 5432 433
5433 5433 434
5434 5434 435
5435 5435 436
5436 5436 437
5437 5437 438
5438 5438 439
5439 5439 440
5440 5440 441
5441 5441 442
5442 5442 443
5443 5443 444
5444 5444 445
5445 5445 446
5446 5446 447
5447 5447 448
5448 5448 449
5449 5449 450
5450 5450 451
5451 5451 452
5452 5452 453
5453 5453 454
5454 5454 455
5455 5455 456
5456 5456 457
5457 5457 458
5458 5458 459
5459 5459 460
5460 5460 461
5461 5461 462
5462 5462 463
5463 5463 464
5464 5464 465
5465 5465 466
5466 5466 467
5467 5467 468
5468 5468 469
5469 5469 470
5470 5470 471
5471 5471 472
5472 5472 473
5473 5473 474
5474 5474 475
5475 5475 476
5476 5476 477
5477 5477 478
5478 5478 479
5479 5479 480
5480 5480 481
5481 5481 482
5482 5482 483
5483 5483 484
5484 5484 485
5485 5485 486
5486 5486 487
5487 5487 488
5488 5488 489
5489 5489 490
5490 5490 491
5491 5491 492
5492 5492 493
5493 5493 494
5494 5494 495
5495 5495 496
5496 5496 497
5497 5497 498
5498 5498 499
5499 5499 500
5500 5500 501
5501 5501 502
5502 5502 503
5503 5503 504
5504 5504 505
5505 5505 506
5506 5506 507
5507 5507 508
5508 5508 509
5509 5509 510
5510 5510 511
5511 5511 512
5512 5512 513
5513 5513 514
5514 5514 515
5515 5515 516
5516 5516 517
5517 5517 518
5518 5518 519
5519 5519 520
5520 5520 521
5521 5521 522
5522 5522 523
5523 5523 524
5524 5524 525
5525 5525 526
5526 5526 527
5527 5527 528
5528 5528 529
5529 5529 530
5530 5530 531
5531 5531 532
5532 5532 533
5533 5533 534
5534 5534 535
5535 5535 536
5536 5536 537
5537 5537 538
5538 5538 539
5539 5539 540
5540 5540 541
5541 5541 542
5542 5542 543
5543 5543 544
5544 5544 545
5545 5545 546
5546 5546 547
5547 5547 548
5548 5548 549
5549 5549 550
5550 5550 551
5551 5551 552
5552 5552 553
5553 5553 554
5554 5554 555
5555 5555 556
5556 5556 557
5557 5557 558
5558 5558 559
5559 5559 560
5560 5560 561
5561 5561 562
5562 5562 563
5563 5563 564
5564 5564 565
5565 5565 566
5566 5566 567
5567 5567 568
5568 5568 569
5569 5569 570
5570 5570 571
5571 5571 572
5572 5572 573
5573 5573 574
5574 5574 575
5575 5575 576
5576 5576 577
5577 5577 578
5578 5578 579
5579 5579 580
5580 5580 581
5581 5581 582
5582 5582 583
5583 5583 584
5584 5584 585
5585 5585 586
5586 5586 587
5587 5587 588
5588 5588 589
5589 5589 590
5590 5590 591
5591 5591 592
5592 5592 593
5593 5593 594
5594 5594 595
5595 5595 596
5596 5596 597
5597 5597 598
5598 5598 599
5599 5599 600
5600 5600 601
5601 5601 602
5602 5602 603
5603 5603 604
5604 5604 605
5605 5605 606
5606 5606 607
5607 5607 608
5608 5608 609
5609 5609 610
5610 5610 611
5611 5611 612
5612 5612 613
5613 5613 614
5614 5614 615
5615 5615 616
5616 5616 617
5617 5617 618
5618 5618 619
5619 5619 620
5620 5620 621
5621 5621 622
5622 5622 623
5623 5623 624
5624 5624 625
5625 5625 626
5626 5626 627
5627 5627 628
5628 5628 629
5629 5629 630
5630 5630 631
5631 5631 632
5632 5632 633
5633 5633 634
5634 5634 635
5635 5635 636
5636 5636 637
5637 5637 638
5638 5638 639
5639 5639 640
5640 5640 641
5641 5641 642
5642 5642 643
5643 5643 644
5644 5644 645
5645 5645 646
5646 5646 647
5647 5647 648
5648 5648 649
5649 5649 650
5650 5650 651
5651 5651 652
5652 5652 653
5653 5653 654
5654 5654 655
5655 5655 656
5656 5656 657
5657 5657 658
5658 5658 659
5659 5659 660
5660 5660 661
5661 5661 662
5662 5662 663
5663 5663 664
5664 5664 665
5665 5665 666
5666 5666 667
5667 5667 668
5668 5668 669
5669 5669 670
5670 5670 671
5671 5671 672
5672 5672 673
5673 5673 674
5674 5674 675
5675 5675 676
5676 5676 677
5677 5677 678
5678 5678 679
5679 5679 680
5680 5680 681
5681 5681 682
5682 5682 683
5683 5683 684
5684 5684 685
5685 5685 686
5686 5686 687
5687 5687 688
5688 5688 689
5689 5689 690
5690 5690 691
5691 5691 692
5692 5692 693
5693 5693 694
5694 5694 695
5695 5695 696
5696 5696 697
5697 5697 698
5698 5698 699
5699 5699 700
5700 5700 701
5701 5701 702
5702 5702 703
5703 5703 704
5704 5704 705
5705 5705 706
5706 5706 707
5707 5707 708
5708 5708 709
5709 5709 710
5710 5710 711
5711 5711 712
5712 5712 713
5713 5713 714
5714 5714 715
5715 5715 716
5716 5716 717
5717 5717 718
5718 5718 719
5719 5719 720
5720 5720 721
5721 5721 722
5722 5722 723
5723 5723 724
5724 5724 725
5725 5725 726
5726 5726 727
5727 5727 728
5728 5728 729
5729 5729 730
5730 5730 731
5731 5731 732
5732 5732 733
5733 5733 734
5734 5734 735
5735 5735 736
5736 5736 737
5737 5737 738
5738 5738 739
5739 5739 740
5740 5740 741
5741 5741 742
5742 5742 743
5743 5743 744
5744 5744 745
5745 5745 746
5746 5746 747
5747 5747 748
5748 5748 749
5749 5749 750
5750 5750 751
5751 5751 752
5752 5752 753
5753 5753 754
5754 5754 755
5755 5755 756
5756 5756 757
5757 5757 758
5758 5758 759
5759 5759 760
5760 5760 761
5761 5761 762
5762 5762 763
5763 5763 764
5764 5764 765
5765 5765 766
5766 5766 767
5767 5767 768
5768 5768 769
5769 5769 770
5770 5770 771
5771 5771 772
5772 5772 773
5773 5773 774
5774 5774 775
5775 5775 776
5776 5776 777
5777 5777 778
5778 5778 779
5779 5779 780
5780 5780 781
5781 5781 782
5782 5782 783
5783 5783 784
5784 5784 785
5785 5785 786
5786 5786 787
5787 5787 788
5788 5788 789
5789 5789 790
5790 5790 791
5791 5791 792
5792 5792 793
5793 5793 794
5794 5794 795
5795 5795 796
5796 5796 797
5797 5797 798
5798 5798 799
5799 5799 800
5800 5800 801
5801 5801 802
5802 5802 803
5803 5803 804
5804 5804 805
5805 5805 806
5806 5806 807
5807 5807 808
5808 5808 809
5809 5809 810
5810 5810 811
5811 5811 812
5812 5812 813
5813 5813 814
5814 5814 815
5815 5815 816
5816 5816 817
5817 5817 818
5818 5818 819
5819 5819 820
5820 5820 821
5821 5821 822
5822 5822 823
5823 5823 824
5824 5824 825
5825 5825 826
5826 5826 827
5827 5827 828
5828 5828 829
5829 5829 830
5830 5830 831
5831 5831 832
5832 5832 833
5833 5833 834
5834 5834 835
5835 5835 836
5836 5836 837
5837 5837 838
5838 5838 839
5839 5839 840
5840 5840 841
5841 5841 842
5842 5842 843
5843 5843 844
5844 5844 845
5845 5845 846
5846 5846 847
5847 5847 848
5848 5848 849
5849 5849 850
5850 5850 851
5851 5851 852
5852 5852 853
5853 5853 854
5854 5854 8
```


APPENDIX B HSO AND A TO D UNDER INTERRUPT CONTROL

SERIES-III MCS-96 MACRO ASSEMBLER, V1 0

SOURCE FILE: F3:A2DHSO.A96

OBJECT FILE: F3:A2DHSO.OBJ

CONTROLS SPECIFIED IN INVOCATION COMMAND: NOSB

ERR LOC	OBJECT	LINE	SOURCE STATEMENT
		1	\$TITLE ('A2DHSO.A96: GENERATING PWM OUTPUTS FROM A TO D INPUTS')
		2	
		3	; This program will provide 3 PWM outputs on HSO pins 0-2
		4	; and one on the PWM.
		5	
		6	; The PWM values are determined by the input to the A/D converter.
		7	
		8	;;;
		9	
		10	\$INCLUDE(DEMO96.INC)
=1		11	\$nolist ; Turn listing off for include file
=1		59	; End of include file
		60	
002B		61	RSEG AT 28H
		62	
001E		63	DL EQU DX: BYTE
		64	
002B		65	ON_TIME:
002B		66	PWM_TIME_1: DSW 1
002A		67	HSO_ON_0: DSW 1
002C		68	HSO_ON_1: DSW 1
002E		69	HSO_ON_2: DSW 1
		70	
0030		71	RESULT_TABLE:
0030		72	RESULT_0: DSW 1
0032		73	RESULT_1: DSW 1
0034		74	RESULT_2: DSW 1
0036		75	RESULT_3: DSW 1
		76	
003B		77	NXT_ON_T: DSW 1
003A		78	NXT_OFF_0: DSW 1
003C		79	NXT_OFF_1: DSW 1
003E		80	NXT_OFF_2: DSW 1
0040		81	COUNT: DSL 1
0044		82	AD_NUM: DSW 1 Channel being converted
0046		83	TMP: DSW 1
0048		84	HSO_PER: DSW 1
004A		85	LAST_LOAD: DSW 1
		86	

310061-86

270061-87

```

2000      87      cseg      AT 2000H
2000 8020      88
2002 1D21      89      DCW      start      ; Timer_ovf_int
2004 8020      90      DCW      Atod_done_int
2006 CC20      91      DCW      start      ; HSI_data_int
2006      92      DCW      HSD_exec_int
2006      93
2006      94      $EJECT
2006      95
2080      96      cseg      AT 2080H
2080      97
2080 A100011B  98      start: LD      SP, #100H      ; Set Stack Pointer
2084 011C      99      CLR      AX
2086 051C      100     wait: DEC      AX      ; wait approx. 0.2 seconds for
2088 D7FC      101     JNE      wait      ; SBE to finish communications
2088      102
208A 1144      103     CLR      AD_NUM
208A      104
208C A180002B  105     LD      PWM_TIME_1, #0B0H
2090 A100014B  106     LD      HSD_PER, #100H
2094 A140002A  107     LD      HSD_ON_0, #040H
2098 A180002C  108     LD      HSD_ON_1, #0B0H
209C A1C0002E  109     LD      HSD_ON_2, #0C0H
209C      110
20A0 4500010A3B 111     ADD      NXT_ON_T, Timer1, #100H
20A0      112
20A5 B13606      113     LDB      HSD_COMMAND, #00110110B ; Set HSD for timer1, set pin 0,1
20AB A03B04      114     LD      HSD_TIME, NXT_ON_T ; with interrupt
20AB FD      115     NOP
20AC FD      116     NOP
20AD B12206      117     LDB      HSD_COMMAND, #00100010B ; Set HSD for timer1, set pin 2
20B0 643B04      118     ADD      HSD_TIME, NXT_ON_T ; without interrupt
20B0      119
20B3 91074A      120     ORB      LAST_LOAD, #00000111B ; Last loaded value was set all pins
20B6 B10A0B      121     LDB      INT_MASK, #00001010B ; Enable HSD and A/D interrupts
20B9 B10A09      122     LDB      INT_PENDING, #00001010B ; Fake an A/D and HSD interrupt
20BC FB      123     EI
20BC      124
20BD 91010F      125     loop: ORB      Port1, #00000001B ; set P1.0
20C0 65010040  126     ADD      COUNT, #01
20C4 A40042      127     ADDC     COUNT+2, zero
20C7 71FE0F      128     ANDB     Port1, #111111110B ; clear P1.0
20CA 27F1      129     BR      loop
20CA      130
20CA      131     $EJECT
20CA      132
20CC      133
20CC      134
20CC      135

```

270061-88

```

132
133 .....
134 ..... HSO EXECUTED INTERRUPT .....
135 .....
136
20CC 137 HSO_exec_int
20CC F2 138 PUSHF
20CD 91020F 139 ORB Port1, #00000010B ; Set p1.1
140
20D0 48380A46 141 SUB TMP, TIMER1, NXT_ON_T
20D4 880046 142 CMP TMP, ZERO
20D7 DE19 143 JLT set_off_times
144
20D9 145 set_off_times:
20D9 644838 146 ADD NXT_ON_T, HSO_PER
20DC B13606 147 LDB HSO_COMMAND, #00110110B ; Set HSO for timer1, set pin 0,1
20DF A03804 148 LD HSO_TIME, NXT_ON_T
20E2 FD 149 NOP
20E3 FD 150 NOP
20E4 B12206 151 LDB HSO_COMMAND, #00100010B ; Set HSO for timer1, set pin 2
20E7 A03804 152 LD HSO_TIME, NXT_ON_T
153
20EA 91074A 154 ORB LAST_LOAD, #00000111B ; Last loaded value was all ones
155
20ED B02817 156 LDB PWM_CONTROL, PWM_TIME_1 ; Now is as good a time as any
157 ; to update the PWM reg
20F0 2026 158 BR check_done
159
20F2 160
20F2 304A23 161 set_off_times:
20F2 162 JBC LAST_LOAD, 0, check_done
163
20F5 442A383A 164 ADD NXT_OFF_0, NXT_ON_T, HSO_ON_0
20F9 B11006 165 LDB HSO_COMMAND, #00010000B ; Set HSO for timer1, clear pin 0
20FC A03A04 166 LD HSO_TIME, NXT_OFF_0
167
20FF FD 168 NOP
2100 442C383C 169 ADD NXT_OFF_1, NXT_ON_T, HSO_ON_1
2104 B11106 170 LDB HSO_COMMAND, #00010001B ; Set HSO for timer1, clear pin 1
2107 A03C04 171 LD HSO_TIME, NXT_OFF_1
172
210A FD 173 NOP
210B 442E383E 174 ADD NXT_OFF_2, NXT_ON_T, HSO_ON_2
210F B11206 175 LDB HSO_COMMAND, #00010010B ; Set HSO for timer1, clear pin 2
2112 A03E04 176 LD HSO_TIME, NXT_OFF_2
177
2115 71F84A 178 ANDB LAST_LOAD, #11111000B ; Last loaded value was all 0s
179
2118 180 check_done:
2118 71FDOF 181 ANDB Port1, #1111101B ; Clear P1.1
182

```

530001-88

270061-89

СХИМЕНА ПОД ЗАПИСИ

270061-90

```

ASSEMBLY COMPLETED, NO ERROR(S) FOUND:  EPP 0049  MOD 2 12 0000 104 04000000 0000 0001 5 12 0000 104
                                     1012 000000 0000000 0 0000000 000000000000 00000000 000000 0000
                                     5
                                     1
                                     00000000000000000000 00000000000000000000 00000000000000000000
EPP 100 000001 1000 00000000000000000000 00000000000000000000 00000000000000000000 00000000000000000000
                                     00000000000000000000 00000000000000000000 00000000000000000000
COMPILE SPECIFIED IN INVOCATION COMMAND MOD0
000001 LIFE: 00 000001 000
000000 LIFE: 00 000001 000
RESTER-111 MOD-00 INVOC0 APPROPRIATE AT 0

```

APPENDIX C SOFTWARE SERIAL PORT

SERIES-III MCS-96 MACRO ASSEMBLER, V1.0

SOURCE FILE: F3 SWPORT.A96

OBJECT FILE: F3 SWPORT.OBJ

CONTROLS SPECIFIED IN INVOCATION COMMAND: NOSB

```

ERR LOC OBJECT LINE SOURCE STATEMENT
1 $TITLE('SWPORT.A96 : SOFTWARE IMPLEMENTED ASYNCHRONOUS SERIAL PORT')
2
3 ; This module provides a software implemented asynchronous serial port
4 ; for the 8096. HSO.5 is used for transmit data. HSI.2 is used for
5 ; receive data. Note: the choice of HSO.5 and HSI.2 is arbitrary).
6
7 $INCLUDE(Demo96.INC)
8 $nolist ; Turn listing off for include file
9
10 ; End of include file
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1090
1091
1092
1093
1094
1095
1096
1097
1098
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1189
1190
1191
1192
1193
1194
1195
1196
1197
1198
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1290
1291
1292
1293
1294
1295
1296
1297
1298
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2260
2261
2262
2263
2264
2265
2266
2267
2268
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2279
2280
2281
2282
2283
2284
2285
2286
2287
2288
2289
2290
2291
2292
2293
2294
2295
2296
2297
2298
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2319
2320
2321
2322
2323
2324
2325
2326
2327
2328
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2339
2340
2341
2342
2343
2344
2345
2346
2347
2348
2349
2350
2351
2352
2353
2354
2355
2356
2357
2358
2359
2360
2361
2362
2363
2364
2365
2366
2367
2368
2369
2370
2371
2372
2373
2374
2375
2376
2377
2378
2379
2380
2381
2382
2383
2384
2385
2386
2387
2388
2389
2390
2391
2392
2393
2394
2395
2396
2397
2398
2399
2400
2401
2402
2403
2404
2405
2406
2407
2408
2409
2410
2411
2412
2413
2414
2415
2416
2417
2418
2419
2420
2421
2422
2423
2424
2425
2426
2427
2428
2429
2430
2431
2432
2433
2434
2435
2436
2437
2438
2439
2440
2441
2442
2443
2444
2445
2446
2447
2448
2449
2450
2451
2452
2453
2454
2455
2456
2457
2458
2459
2460
2461
2462
2463
2464
2465
2466
2467
2468
2469
2470
2471
2472
2473
2474
2475
2476
2477
2478
2479
2480
2481
2482
2483
2484
2485
2486
2487
2488
2489
2490
2491
2492
2493
2494
2495
2496
2497
2498
2499
2500
2501
2502
2503
2504
2505
2506
2507
2508
2509
2510
2511
2512
2513
2514
2515
2516
2517
2518
2519
2520
2521
2522
2523
2524
2525
2526
2527
2528
2529
2530
2531
2532
2533
2534
2535
2536
2537
2538
2539
2540
2541
2542
2543
2544
2545
2546
2547
2548
2549
2550
2551
2552
2553
2554
2555
2556
2557
2558
2559
2560
2561
2562
2563
2564
2565
2566
2567
2568
2569
2570
2571
2572
2573
2574
2575
2576
2577
2578
2579
2580
2581
2582
2583
2584
2585
2586
2587
2588
2589
2590
2591
2592
2593
2594
2595
2596
2597
2598
2599
2600
260
```

```

2080      88      cseg at 2080h
2080      89
2080      90      reset_loc:
2080      91      ; The 8096 starts executing here on reset, the program will initialize the
2080      92      ; the software serial port and run a simple test to exercise it.
2080      93
2080      94      di
2081      95      ld      sp, #0f0h
2085      96      push   #4800
2088      97      call   setup_serial_port
2088      98      ldb     int_mask, #01101100b
2088      99      ei
2088      100
2088      101
2088      102      test1:
2088      103      ; A simple test of the serial port routines.
2088      104      ; While no characters are received an incrementing pattern is sent to the
2088      105      ; serial output. When a character is received the incrementing pattern
2088      106      ; "jumps" to the character received and proceeds from there.
2088      107
2088      108      CR      equ     ODH
2088      109      ldb     char, #CR
2088      110      testiloop:
2088      111      ldbne   ax, char
2088      112      push   ax
2088      113      call   char_out
2088      114
2088      115      cmpb    char, #CR
2088      116      bne     nopause
2088      117      clr     ax
2088      118      pause:
2088      119      inc     ax
2088      120      bne     pause
2088      121      nopause:
2088      122
2088      123      incb    char
2088      124      test2:
2088      125      call   csts
2088      126      cmpb    al, 0
2088      127      be      testiloop
2088      128      call   char_in
2088      129      ldb     char, al
2088      130      br     testiloop
2088      131      #eject
2088      132
2088      133
2088      134
2088      135

```

270061-92

```

0000      132
133      133      cseg
134      134
0000      135      setup_serial_port
136      136      ; Called on system reset to initiate the software serial port
137      137
0000 CC22      138      pop      cx      ; the return address
0002 CC20      139      pop      bx      ; the baud rate (in decimal)
0004 A107001E 140      ld      dx, #0007h      ; dx:ax =500,000 (assumes 12 Mhz crystal)
0008 A120A11C 141      ld      ax, #0A120h
000C BC201C      142      divu     ax, bx      ; calculate the baud count (500,000/baudrate)
000F C0081C      143      st      ax, baud_count
0012 C00600      144      st      0, serial_out      ; clear serial out
0015 B16016      145      ldb     ioctl, #01100000b      ; Enable HSO.5 and Txd
0018 3E15FD      146      bbs     ios0.6, $      ; Wait for room in the HSO CAM
147      147      ; and issue a MARK command.
001B 44140A0A      148      add     txd_time, timer1, 20
001F B13506      149      ldb     hso_command, #mark_command
0022 A00A04      150      ld      hso_time, txd_time
0025 1102      151      clrb     rcv_buf      ; clear out the receive variables
0027 1103      152      clrb     rcv_reg
0029 1101      153      clrb     rcv_state
002B EF4800      154      call    init_receive      ; setup to detect a start bit
002E E322      155      br      [cx]      ; return
0030 680B0C      156      $reject
157      157
0030 Eb3000      158      char_out
159      159      ; Output character to the software serial port
160      160
0030 CC22      161      pop      cx      ; the return address
0032 CC20      162      pop      bx      ; the character for output
0034 B10121      163      ldb     (bx+1), #01h      ; add the start and stop bits
0037 642020      164      add     bx, bx      ; to the char and leave as 16 bit
003A      165      wait_for_xmit
003A 880006      166      cmp     serial_out, 0      ; wait for serial_out=0 (it will be cleared by
003D D7FB      167      bne     wait_for_xmit      ; the hso interrupt process)
003F C00620      168      st      bx, serial_out      ; put the formatted character in serial_out
0042 E322      169      br      [cx]      ; return to caller
170      170
0044      171      csts:
172      172      ; Returns "true" (ax<>0) if char_in has a character
173      173
0044 011C      174      clr     ax
0046 300102      175      bbc     rcv_state, 0, csts_exit
0049 071C      176      inc     ax
004B      177      csts_exit:
004B F0      178      ret
179      179
004C      180      char_in
181      181

```

270061-93


```

181 ; Get a character from the software serial port
182
183 ; wait for character ready
184 bbc rcv_state,0,char_in
185 pushf ; set up a critical region
186 andb rcv_state,#not(rxrdy)
187 ldbz al,rcv_buf
188 popf ; leave the critical region
189 ret
190 $eject
191
192 hso_isr
193 ; Fields the hso interrupts and performs the serialization of the data
194 ; Note this routine would be incorporated into the hso service strategy
195 ; for an actual system
196
197 cseg at 2006h
198 dcw hso_isr ; Set up vector
199
200 cseg
201 pushf
202 add txd_time,baud_count
203 cmp serial_out,0 ; if character is done send a mark
204 be send_mark
205 shr serial_out,#1 ; else send bit 0 of serial_out and shift
206 bc send_mark ; serial_out left one place.
207 send_space:
208 ldb hso_command,#space_command
209 ld hso_time,txd_time
210 br hso_isr_exit
211 send_mark:
212 ldb hso_command,#mark_command
213 ld hso_time,txd_time
214
215 hso_isr_exit:
216 popf
217 ret
218 $eject
219
220 init_receive:
221 ; Called to prepare the serial input process to find the leading edge of
222 ; a start bit
223
224 ldb ioc0,#00000000b ; disconnect change detector
225 ldb hsi_mode,#00100000b ; negative edges on HSI 2
226 flush_fifo
227 orb ios1_save,ios1
228 bbc ios1_save,7,flush_fifo_done
229 ldb al,hsi_status
230 ld ax,hsi_time ; trash the fifo entry

```

```

0088 717F00      R    231      andb    ios1_save,#not(80h)      ; clear bit 7.
0088 27EF        232      br        flush_fifo
008D 00041C      233      flush_fifo_done:
008D B11015      234      ldb     ioc0,#00010000b      ; connect HSI.2 to detector
0090 F0         235      ret
0090 801F00      236
0090 801F00      237
0090 B15003      238
0091 B10019      239      hsi_isr:
240      ; Fields interrupts from the HSI unit, used to detect the leading edge
241      ; of the START bit
242      ; Note: this routine would be incorporated into the HSI strategy of an actual
243      ; system.
244
2004            245      cseg at 2004h
2004 9100      R    246      dcw     hsi_isr      ; setup the interrupt vector
2004            247
0091            248      cseg
0091 F2        249      pushf
0092 CB1C      250      push    ax
0094 B0061C      251      ldb     al,hsi_status
0097 A00404      R    252      ld      sample_time,hsi_time
009A 341C15      253      bbc     al,4,exit_hsi
009D 3F15FD      254      bbs     ios0,7,$      ; wait for room in HSD holding reg
00A0 A00B1C      R    255      ld      ax,baud_count      ; send out sample command in 1/2
00A3 0B011C      256      shr     ax,#1      ; bit time
00A6 641C04      R    257      add     sample_time,ax
00A9 B11806      258      ldb     hso_command,#sample_command
00AC C00404      R    259      st      sample_time,hso_time
00AF B10015      260      ldb     ioc0,#00000000b      ; disconnect hsi.2 from change detector
00B2            261      exit_hsi:
00B2 CC1C      262      pop     ax
00B4 F3        263      popf
00B5 F0        264      ret
00B5            265      $elect
00B5            266
00B6            267      software_timer_isr:
268      ; Fields the software timer interrupt, used to deserialize the incoming data
269      ; Note this routine would be incorporated into the software timer strategy
270      ; in an actual system.
271
200A            272      cseg at 200ah
200A B600      R    273      dcw     software_timer_isr      ; setup vector
200A            274
00B6            275      cseg
00B6 F2        276      pushf
00B7 901600      R    277      orb     ios1_save,ios1
00BA 71FE00      278      andb    ios1_save,#not(01h)      ; clear bit 0
00BD 51FC0100    R    279      andb    0,rcv_state,#0fch      ; All bits except rxrdy and overrun=0
00C1 D70C      280      bne     process_data

```

270061-95

```

00C3          281  process_start_bit:
00C3 350604    282      bbc      hsi_status,5,start_ok
00C6 2FAE     283      call     init_receive
00C8 2032     284      br       software_timer_exit
00CA          285  start_ok:
00CA 910401    R 286      orb      rcve_state,#rip ; set receive in progress flag
00CD 2021     287      br       schedule_sample
00CE          288
00CF          289  process_data:
00CF 3F010E    R 290      bbs      rcve_state,7,check_stopbit
00D2 180103    R 291      shrb     rcve_reg,#1
00D5 350603    292      bbc      hsi_status,5,datazero
00D8 918003    R 293      orb      rcve_reg,#80h ; set the new data bit
00DB          294  datazero:
00DB 751001    R 295      addb     rcve_state,#10h ; increment bit count
00DE 2010     296      br       schedule_sample
00E0          297
00E0          298  check_stopbit:
00E0 3506FD    299      bbc      hsi_status,5,$ ; DEBUG ONLY
00E3 800302    R 300      ldb      rcve_buf,rcve_reg
00E6 910101    R 301      orb      rcve_state,#rirdy
00E9 710301    R 302      andb     rcve_state,#03h ; Clear all but ready and overrun bits
00EC 2FB8     303      call     init_receive
00EE 200C     304      br       software_timer_exit
00F0          305
00F0          306  schedule_sample:
00F0 3F15FD    307      bbs      ios0,7,$ ; wait for holding reg empty
00F3 B11806    308      ldb      hso_command,#sample_command
00F6 640804    R 309      add      sample_time,baud_count
00F9 C00404    R 310      st       sample_time,hso_time
00FC          311
00FC          312  software_timer_exit:
00FC          313      popf
00FD F0       314      ret
00FE          315
00FE          316
00FE          317  end

```

ASSEMBLY COMPLETED. - NO ERROR(S) FOUND

270061-96

intel

AP-248

intel

SERIES-III MCS-96 MACRO ASSEMBLER, V1 0

SOURCE FILE: :F3:MOTCON.A96

OBJECT FILE: :F3:MOTCON.OBJ

CONTROLS SPECIFIED IN INVOCATION COMMAND: NOSB

ERR LOC	OBJECT	LINE	SOURCE STATEMENT
		1	\$TITLE ('MOTCON.A96: Motor Control Example Program')
		2	
		3	USE WITH C-STEP or later parts
		4	
		5	December 20, 1984
		6	
		7	\$INCLUDE(DEMO96.INC)
		8	\$nolist ; Turn listing off for include file
		9	; End of include file
		10	
		11	Initial Values
		12	
		13	min_hsil_t equ 30 ; min period for PHA edges in model before mode2
		14	
		15	min_hsi_t equ 2*min_hsil_t ; min period for PHA edges in mode0 before model
		16	
		17	max_hsil_t equ 3*min_hsil_t + min_hsil_t/2
		18	; max period for PHA edges in model before mode0
		19	
		20	HSD0_dly_period equ 110 ; delay for HSD timer 0 (timed count of pulses)
		21	; min period for 5 T2 clocks before mode 1
		22	
		23	swt1_dly_period equ 250 ; delay for software timer 1
		24	swt2_dly_period equ 250 ; delay for software timer 2
		25	max_power equ 0ffh
		26	max_brake equ 0ffh
		27	maximum_hold equ 080h
		28	brake_pnt equ 1200
		29	position_pnt equ 100
		30	velocity_pnt equ 16
		31	
		32	RSEQ at 024H
		33	
		34	tmp: dsl 1
		35	timer_2: dsl 1
		36	
		37	
		38	
		39	
		40	
		41	
		42	
		43	
		44	
		45	
		46	
		47	
		48	
		49	
		50	
		51	
		52	
		53	
		54	
		55	
		56	
		57	
		58	
		59	
		60	
		61	
		62	
		63	
		64	
		65	
		66	
		67	
		68	
		69	
		70	
		71	
		72	
		73	
		74	
		75	
		76	
		77	
		78	
		79	
		80	
		81	
		82	
		83	
		84	
		85	

270061-97

APPENDIX D MOTOR CONTROL PROGRAM

530021-00

```

002C      86      tmr2_old:      dsl 1
0030      87      position:     dsl 1
0034 816110      88      des_pos:  b0165' dsl 1
0038 816106      89      pos_err: b0167' dsl 1
003C      90      delta_p:      dsl 1
0040 054F      91      time:      dsl 1
0044 88003C      92      des_time: b0167' dsl 1
0048 80986D      93      time_err: b0167' dsl 1
0050 032C      94      $EJECT  19      pwb1: 80000
0054 810113C      95      last_time_err dsw 1
0058 119B      96      last_pos_err dsw 1
004C      97      pos_delta: dsw 1
004E 816115      98      time_delta dsw 1
0050 816001B      99      last_pos dsw 1
0052      100      last1_time: dsw 1
0054      101      last2_time: dsw 1
0056      102      boost:      dsw 1
0058      103      tmp1:        dsw 1
005A      104      out_ptr:     dsw 1
005C      105      offset:      dsw 1
005E      106      nxt_pos:     dsw 1
0060      107      npwr:        dsw 1
0062 1050      108      old_t2: 800F' dsw 1
0064 1050      109      direct: p00' dsw 1 ; 1=forward, 0=reverse
0066 5055      110      pwm_dir: p00' dsw 1
0068 1050      111      hsi_s0: 800F' dsw 1
006A 0055      112      last_stat: 800F' dsw 1
006C      113      pwm_pwr: 5000H dsw 1
006D      114      ios1_bak: dsw 1
006E      115      TR_COL: DSB 1 COLLECT TRACE IF TR_COL=00
006F      116      main_dly: 800F' dsw 1
0070      117      3a      b1 1 dsw 1
0072      118      max_pwr: b1 0 dsw 1
0074      119      max_brk: b1 0 dsw 1
0076      120      max_hold: b1 0 dsw 1
0078      121      vel_pnt: b1 3 dsw 1
007A      122      brk_pnt: b1 3 dsw 1
007C      123      pos_pnt: b1 1 dsw 1
007E      124      HSOO_dly: b1 0 dsw 1
0080      125      swt1_dly: b1 0 dsw 1
0082      126      swt2_dly: b1 0 dsw 1
0084      127      min_hsi: dsw 1
0086      128      min_hsi1: dsw 1
0088      129      max_hsi1: dsw 1
0090      130      dseg at 100H
0092      131      3a      b1 1 dsw 1
0094      132      3a      b1 1 dsw 1
0096      133      3a      b1 1 dsw 1
0098      134      3a      b1 1 dsw 1
009A      135      3a      b1 1 dsw 1
009C      136      3a      b1 1 dsw 1
009E      137      3a      b1 1 dsw 1
00A0      138      3a      b1 1 dsw 1
00A2      139      3a      b1 1 dsw 1
00A4      140      3a      b1 1 dsw 1
00A6      141      3a      b1 1 dsw 1
00A8      142      3a      b1 1 dsw 1
00AA      143      3a      b1 1 dsw 1
00AC      144      3a      b1 1 dsw 1
00AE      145      3a      b1 1 dsw 1
00B0      146      3a      b1 1 dsw 1
00B2      147      3a      b1 1 dsw 1
00B4      148      3a      b1 1 dsw 1
00B6      149      3a      b1 1 dsw 1
00B8      150      3a      b1 1 dsw 1
00BA      151      3a      b1 1 dsw 1
00BC      152      3a      b1 1 dsw 1
00BE      153      3a      b1 1 dsw 1
00C0      154      3a      b1 1 dsw 1
00C2      155      3a      b1 1 dsw 1
00C4      156      3a      b1 1 dsw 1
00C6      157      3a      b1 1 dsw 1
00C8      158      3a      b1 1 dsw 1
00CA      159      3a      b1 1 dsw 1
00CC      160      3a      b1 1 dsw 1
00CE      161      3a      b1 1 dsw 1
00D0      162      3a      b1 1 dsw 1
00D2      163      3a      b1 1 dsw 1
00D4      164      3a      b1 1 dsw 1
00D6      165      3a      b1 1 dsw 1
00D8      166      3a      b1 1 dsw 1
00DA      167      3a      b1 1 dsw 1
00DC      168      3a      b1 1 dsw 1
00DE      169      3a      b1 1 dsw 1
00E0      170      3a      b1 1 dsw 1
00E2      171      3a      b1 1 dsw 1
00E4      172      3a      b1 1 dsw 1
00E6      173      3a      b1 1 dsw 1
00E8      174      3a      b1 1 dsw 1
00EA      175      3a      b1 1 dsw 1
00EC      176      3a      b1 1 dsw 1
00EE      177      3a      b1 1 dsw 1
00F0      178      3a      b1 1 dsw 1
00F2      179      3a      b1 1 dsw 1
00F4      180      3a      b1 1 dsw 1
00F6      181      3a      b1 1 dsw 1
00F8      182      3a      b1 1 dsw 1
00FA      183      3a      b1 1 dsw 1
00FC      184      3a      b1 1 dsw 1
00FE      185      3a      b1 1 dsw 1
00FF      186      3a      b1 1 dsw 1

```

270061-98

```

136
0100 137 mode_view      dsb      1
0102 138 count_out      dsw      1
0104 139 err_view      dsw      1
140
141
142 $eject          wdt_batt:   qm 1
143
144 PIN#            PORT      FLAG_USAGE
145
146 22 P1 0          mode0 0    mode1 1    mode2 1    or 0
147 23 P1 1          0        0        1        1
148 24 P1 2          software timer 2 routine enter/leave
149 25 P1 3          Main program toggle
150 26 P1 4          HSI overflow toggle
151 37 P1 5          software timer 0 routine enter/leave
152 38 P1 6          hsi_int enter/leave
153 39 P1 7          software timer 1 routine enter/leave
154 40 P2 6          Input direction (0=reverse, 1=forward)
155 45 P2 7          direction 0=rev, 1=fwd
156
157 cseg            at 2000H
158 dcw timer_ovf_int
159 dcw atod_done_int
160 dcw hsi_data_int
161 dcw hso_exec_int
162 dcw hsi_0_int
163 dcw soft_tmr_int
164 dcw ser_port_int
165 dcw external_int
166
167 atod_done_int:
168 hsi_0_int:
169 ser_port_int:
170 external_int:
171
172 cseg            at 2080H
173
174 init:           ld sp, #0FOH
175 ldb pwm_control, #OFFH
176
177 clrb direct
178 ld tmp1, #6000 ; wait about 3 seconds for motor
179 delay:          dec tmp1 ; to come to a stop
180 djnz direct, $ ; wait 0.512 milliseconds
181 cmp tmp1, zero
182 jgt delay
183
184 ldb port1, #OFFH
185 ldb port2, #OFFH
186
187
188

```

```

209D B12516      186      ldb      IOC1,#00100101B ; Disable HSD.4,HSD.5, HSI_INT=first,
209E B12517      187      sbc      IOC1,#00100101B ; Enable PWM,TXD,TIMER1_OVRFLOW_INT
209F B12518      188      sbc      IOC1,#00100101B
20A0 71FC0F      189      andb     Port1,#11111100B ; clear P1.0,1 (set mode 0)
20A3 B19903      190      ldb      HSI_mode,#10011001B ; set hsi.1,3 -; hsi.0,2 +
20A6 B13715      191      ldb      IOC0,#01010111B ; Enable all hsi.0,1,2,3
20A9 B19903      192      sbc      IOC0,#01010111B ; T2 CLOCK=T2CLK, T2RST=T2RST
20AC B19903      193      sbc      IOC0,#01010111B ; Clear timer2
20AD B19903      194      sje     $,exit
20AE B19903      195      sbc      IOC1,#00100101B
20AF B19903      196      sbc      IOC1,#00100101B
20B0 A00400      197      clr      zero,hsi_time
20B3 0140        198      clr      time
20B6 0142        199      clr      time+2
20B9 0128        200      clr      timer_2
20BC 012A        201      clr      timer_2+2
20BF 0130        202      clr      position
20C2 0132        203      clr      position+2
20C5 0154        204      clr      last_pos
20C8 0134        205      clr      des_pos
20CB 0136        206      clr      des_pos+2
20CE 0144        207      clr      des_time
20D1 0146        208      clr      des_time+2
20D4 A00A56      209      ld       last1_time,Timer1
20D7 4900085658  210      sub     last2_time,last1_time,#800H
20DA 116D        211      clrb     ios1_bak
20DD 1109        212      clrb     int_pending
20E0 A1F0015E      213      ld       out_ptr,#1FOH
20E3 A13C00B2      214      ld       min_hsi,#min_hsi_t
20E6 A11E00B4      215      ld       min_hsi1,#min_hsi1_t
20E9 A16900B6      216      ld       max_hsi1,#max_hsi1_t
20EC A16E007C      217      ld       HSD0_dly,#HSD0_dly_period
20EF A1FA007E      218      ld       swt1_dly,#swt1_dly_period
20F2 A1FA00B0      219      ld       swt2_dly,#(swt2_dly_period)
20F5 A1FF0070      220      ld       max_pwr,#max_power
20F8 A1FF0072      221      ld       max_brk,#max_brake
20FB A1800074      222      ld       max_hold,#maximum_hold
20FE A1800478      223      ld       brk_pnt,#brake_pnt
20A1 A164007A      224      ld       pos_pnt,#position_pnt
20A4 A1100076      225      ld       vel_pnt,#velocity_pnt
20A7 A1002962      226      ld       nxt_pos,#pos_table
20AA B0006C        227      ldb      pwm_pwr,zero
20AD B10169      228      ldb      pwm_dir,#01h ; FORWARD
20B0 B12D08      229      ldb      int_mask,#00101101B ; Enable tmr_ovf, hsi, swt, HSD.interrupts
20B3 B13006      230      ldb      hso_command,#30H ; set HSD_0
20B6 447C0A04      231      add     hso_time,timer1,HSD0_dly
20B9 FD          232      nop
20BC FD          233      NOP
20BF B13906      234      ldb      hso_command,#39H ; set swt_1
20C2 447E0A04      235      add     hso_time,timer1,swt1_dly

```

270061-A0

530081-V0

```

211F FD          236      nop
2120 FDE0V04    237      nop
2121 B13A06     238      ldb com hso_command,#3AH
2124 44B00A04    239      add hso_time.timer1,swt2_dly
2128 A00A40     240      nop
212B A00G2C     241      ldr time,TIMER1
212E FB5D08     242      ldr com tmr2_old,timer2
212F          243      elc
2130          244      brne q11,main_prog
2131          245      jmp
2132          246      $reject
2133          247      jmp
2134          248      jmp
2135          249      jmp
2136          250      jmp
2137          251      jmp
2138          252      jmp
2139          253      jmp
213A          254      CSEG AT 2200H
213B          255      jmp
213C          256      timer_ovf_int
213D          257      pushf
213E          258      jmp
213F          259      orb
2140          260      chk_t1
2141          261      inc
2142          262      andb
2143          263      tmr_int_done
2144          264      popf
2145          265      ret
2146          266      jmp
2147          267      jmp
2148          268      jmp
2149          269      jmp
214A          270      jmp
214B          271      jmp
214C          272      jmp
214D          273      CSEG AT 2220H
214E          274      jmp
214F          275      jmp
2150          276      soft_tmr_int
2151          277      pushf
2152          278      orb
2153          279      chk_swto
2154          280      jmp
2155          281      andb
2156          282      call
2157          283      chk_swto1
2158          284      jmp
2159          285      andb
2160          286      jmp
2161          287      jmp
2162          288      jmp
2163          289      jmp
2164          290      jmp
2165          291      jmp
2166          292      jmp
2167          293      jmp
2168          294      jmp
2169          295      jmp
2170          296      jmp
2171          297      jmp
2172          298      jmp
2173          299      jmp
2174          300      jmp
2175          301      jmp
2176          302      jmp
2177          303      jmp
2178          304      jmp
2179          305      jmp
2180          306      jmp
2181          307      jmp
2182          308      jmp
2183          309      jmp
2184          310      jmp
2185          311      jmp
2186          312      jmp
2187          313      jmp
2188          314      jmp
2189          315      jmp
2190          316      jmp
2191          317      jmp
2192          318      jmp
2193          319      jmp
2194          320      jmp
2195          321      jmp
2196          322      jmp
2197          323      jmp
2198          324      jmp
2199          325      jmp
2200          326      jmp
2201          327      jmp
2202          328      jmp
2203          329      jmp
2204          330      jmp
2205          331      jmp
2206          332      jmp
2207          333      jmp
2208          334      jmp
2209          335      jmp
2210          336      jmp
2211          337      jmp
2212          338      jmp
2213          339      jmp
2214          340      jmp
2215          341      jmp
2216          342      jmp
2217          343      jmp
2218          344      jmp
2219          345      jmp
2220          346      jmp
2221          347      jmp
2222          348      jmp
2223          349      jmp
2224          350      jmp
2225          351      jmp
2226          352      jmp
2227          353      jmp
2228          354      jmp
2229          355      jmp
2230          356      jmp
2231          357      jmp
2232          358      jmp
2233          359      jmp
2234          360      jmp
2235          361      jmp
2236          362      jmp
2237          363      jmp
2238          364      jmp
2239          365      jmp
2240          366      jmp
2241          367      jmp
2242          368      jmp
2243          369      jmp
2244          370      jmp
2245          371      jmp
2246          372      jmp
2247          373      jmp
2248          374      jmp
2249          375      jmp
2250          376      jmp
2251          377      jmp
2252          378      jmp
2253          379      jmp
2254          380      jmp
2255          381      jmp
2256          382      jmp
2257          383      jmp
2258          384      jmp
2259          385      jmp
2260          386      jmp
2261          387      jmp
2262          388      jmp
2263          389      jmp
2264          390      jmp
2265          391      jmp
2266          392      jmp
2267          393      jmp
2268          394      jmp
2269          395      jmp
2270          396      jmp
2271          397      jmp
2272          398      jmp
2273          399      jmp
2274          400      jmp
2275          401      jmp
2276          402      jmp
2277          403      jmp
2278          404      jmp
2279          405      jmp
2280          406      jmp
2281          407      jmp
2282          408      jmp
2283          409      jmp
2284          410      jmp
2285          411      jmp
2286          412      jmp
2287          413      jmp
2288          414      jmp
2289          415      jmp
2290          416      jmp
2291          417      jmp
2292          418      jmp
2293          419      jmp
2294          420      jmp
2295          421      jmp
2296          422      jmp
2297          423      jmp
2298          424      jmp
2299          425      jmp
2300          426      jmp
2301          427      jmp
2302          428      jmp
2303          429      jmp
2304          430      jmp
2305          431      jmp
2306          432      jmp
2307          433      jmp
2308          434      jmp
2309          435      jmp
2310          436      jmp
2311          437      jmp
2312          438      jmp
2313          439      jmp
2314          440      jmp
2315          441      jmp
2316          442      jmp
2317          443      jmp
2318          444      jmp
2319          445      jmp
2320          446      jmp
2321          447      jmp
2322          448      jmp
2323          449      jmp
2324          450      jmp
2325          451      jmp
2326          452      jmp
2327          453      jmp
2328          454      jmp
2329          455      jmp
2330          456      jmp
2331          457      jmp
2332          458      jmp
2333          459      jmp
2334          460      jmp
2335          461      jmp
2336          462      jmp
2337          463      jmp
2338          464      jmp
2339          465      jmp
2340          466      jmp
2341          467      jmp
2342          468      jmp
2343          469      jmp
2344          470      jmp
2345          471      jmp
2346          472      jmp
2347          473      jmp
2348          474      jmp
2349          475      jmp
2350          476      jmp
2351          477      jmp
2352          478      jmp
2353          479      jmp
2354          480      jmp
2355          481      jmp
2356          482      jmp
2357          483      jmp
2358          484      jmp
2359          485      jmp
2360          486      jmp
2361          487      jmp
2362          488      jmp
2363          489      jmp
2364          490      jmp
2365          491      jmp
2366          492      jmp
2367          493      jmp
2368          494      jmp
2369          495      jmp
2370          496      jmp
2371          497      jmp
2372          498      jmp
2373          499      jmp
2374          500      jmp
2375          501      jmp
2376          502      jmp
2377          503      jmp
2378          504      jmp
2379          505      jmp
2380          506      jmp
2381          507      jmp
2382          508      jmp
2383          509      jmp
2384          510      jmp
2385          511      jmp
2386          512      jmp
2387          513      jmp
2388          514      jmp
2389          515      jmp
2390          516      jmp
2391          517      jmp
2392          518      jmp
2393          519      jmp
2394          520      jmp
2395          521      jmp
2396          522      jmp
2397          523      jmp
2398          524      jmp
2399          525      jmp
2400          526      jmp
2401          527      jmp
2402          528      jmp
2403          529      jmp
2404          530      jmp
2405          531      jmp
2406          532      jmp
2407          533      jmp
2408          534      jmp
2409          535      jmp
2410          536      jmp
2411          537      jmp
2412          538      jmp
2413          539      jmp
2414          540      jmp
2415          541      jmp
2416          542      jmp
2417          543      jmp
2418          544      jmp
2419          545      jmp
2420          546      jmp
2421          547      jmp
2422          548      jmp
2423          549      jmp
2424          550      jmp
2425          551      jmp
2426          552      jmp
2427          553      jmp
2428          554      jmp
2429          555      jmp
2430          556      jmp
2431          557      jmp
2432          558      jmp
2433          559      jmp
2434          560      jmp
2435          561      jmp
2436          562      jmp
2437          563      jmp
2438          564      jmp
2439          565      jmp
2440          566      jmp
2441          567      jmp
2442          568      jmp
2443          569      jmp
2444          570      jmp
2445          571      jmp
2446          572      jmp
2447          573      jmp
2448          574      jmp
2449          575      jmp
2450          576      jmp
2451          577      jmp
2452          578      jmp
2453          579      jmp
2454          580      jmp
2455          581      jmp
2456          582      jmp
2457          583      jmp
2458          584      jmp
2459          585      jmp
2460          586      jmp
2461          587      jmp
2462          588      jmp
2463          589      jmp
2464          590      jmp
2465          591      jmp
2466          592      jmp
2467          593      jmp
2468          594      jmp
2469          595      jmp
2470          596      jmp
2471          597      jmp
2472          598      jmp
2473          599      jmp
2474          600      jmp
2475          601      jmp
2476          602      jmp
2477          603      jmp
2478          604      jmp
2479          605      jmp
2480          606      jmp
2481          607      jmp
2482          608      jmp
2483          609      jmp
2484          610      jmp
2485          611      jmp
2486          612      jmp
2487          613      jmp
2488          614      jmp
2489          615      jmp
2490          616      jmp
2491          617      jmp
2492          618      jmp
2493          619      jmp
2494          620      jmp
2495          621      jmp
2496          622      jmp
2497          623      jmp
2498          624      jmp
2499          625      jmp
2500          626      jmp
2501          627      jmp
2502          628      jmp
2503          629      jmp
2504          630      jmp
2505          631      jmp
2506          632      jmp
2507          633      jmp
2508          634      jmp
2509          635      jmp
2510          636      jmp
2511          637      jmp
2512          638      jmp
2513          639      jmp
2514          640      jmp
2515          641      jmp
2516          642      jmp
2517          643      jmp
2518          644      jmp
2519          645      jmp
2520          646      jmp
2521          647      jmp
2522          648      jmp
2523          649      jmp
2524          650      jmp
2525          651      jmp
2526          652      jmp
2527          653      jmp
2528          654      jmp
2529          655      jmp
2530          656      jmp
2531          657      jmp
2532          658      jmp
2533          659      jmp
2534          660      jmp
2535          661      jmp
2536          662      jmp
2537          663      jmp
2538          664      jmp
2539          665      jmp
2540          666      jmp
2541          667      jmp
2542          668      jmp
2543          669      jmp
2544          670      jmp
2545          671      jmp
2546          672      jmp
2547          673      jmp
2548          674      jmp
2549          675      jmp
2550          676      jmp
2551          677      jmp
2552          678      jmp
2553          679      jmp
2554          680      jmp
2555          681      jmp
2556          682      jmp
2557          683      jmp
2558          684      jmp
2559          685      jmp
2560          686      jmp
2561          687      jmp
2562          688      jmp
2563          689      jmp
2564          690      jmp
2565          691      jmp
2566          692      jmp
2567          693      jmp
2568          694      jmp
2569          695      jmp
2570          696      jmp
2571          697      jmp
2572          698      jmp
2573          699      jmp
2574          700      jmp
2575          701      jmp
2576          702      jmp
2577          703      jmp
2578          704      jmp
2579          705      jmp
2580          706      jmp
2581          707      jmp
2582          708      jmp
2583          709      jmp
2584          710      jmp
2585          711      jmp
2586          712      jmp
2587          713      jmp
2588          714      jmp
2589          715      jmp
2590          716      jmp
2591          717      jmp
2592          718      jmp
2593          719      jmp
2594          720      jmp
2595          721      jmp
2596          722      jmp
2597          723      jmp
2598          724      jmp
2599          725      jmp
2600          726      jmp
2601          727      jmp
2602          728      jmp
2603          729      jmp
2604          730      jmp
2605          731      jmp
2606          732      jmp
2607          733      jmp
2608          734      jmp
2609          735      jmp
2610          736      jmp
2611          737      jmp
2612          738      jmp
2613          739      jmp
2614          740      jmp
2615          741      jmp
2616          742      jmp
2617          743      jmp
2618          744      jmp
2619          745      jmp
2620          746      jmp
2621          747      jmp
2622          748      jmp
2623          749      jmp
2624          750      jmp
2625          751      jmp
2626          752      jmp
2627          753      jmp
2628          754      jmp
2629          755      jmp
2630          756      jmp
2631          757      jmp
2632          758      jmp
2633          759      jmp
2634          760      jmp
2635          761      jmp
2636          762      jmp
2637          763      jmp
2638          764      jmp
2639          765      jmp
2640          766      jmp
2641          767      jmp
2642          768      jmp
2643          769      jmp
2644          770      jmp
2645          771      jmp
2646          772      jmp
2647          773      jmp
2648          774      jmp
2649          775      jmp
2650          776      jmp
2651          777      jmp
2652          778      jmp
2653          779      jmp
2654          780      jmp
2655          781      jmp
2656          782      jmp
2657          783      jmp
2658          784      jmp
2659          785      jmp
2660          786      jmp
2661          787      jmp
2662          788      jmp
2663          789      jmp
2664          790      jmp
2665          791      jmp
2666          792      jmp
2667          793      jmp
2668          794      jmp
2669          795      jmp
2670          796      jmp
2671          797      jmp
2672          798      jmp
2673          799      jmp
2674          800      jmp
2675          801      jmp
2676          802      jmp
2677          803      jmp
2678          804      jmp
2679          805      jmp
2680          806      jmp
2681          807      jmp
2682          808      jmp
2683          809      jmp
2684          810      jmp
2685          811      jmp
2686          812      jmp
2687          813      jmp
2688          814      jmp
2689          815      jmp
2690          816      jmp
2691          817      jmp
2692          818      jmp
2693          819      jmp
2694          820      jmp
2695          821      jmp
2696          822      jmp
2697          823      jmp
2698          824      jmp
2699          825      jmp
2700          826      jmp
2701          827      jmp
2702          828      jmp
2703          829      jmp
2704          830      jmp
2705          831      jmp
2706          832      jmp
2707          833      jmp
2708          834      jmp
2709          835      jmp
2710          836      jmp
2711          837      jmp
2712          838      jmp
2713          839      jmp
2714          840      jmp
2715          841      jmp
2716          842      jmp
2717          843      jmp
2718          844      jmp
2719          845      jmp
2720          846      jmp
2721          847      jmp
2722          848      jmp
2723          849      jmp
2724          850      jmp
2725          851      jmp
2726          852      jmp
2727          853      jmp
2728          854      jmp
2729          855      jmp
2730          856      jmp
2731          857      jmp
2732          858      jmp
2733          859      jmp
2734          860      jmp
2735          861      jmp
2736          862      jmp
2737          863      jmp
2738          864      jmp
2739          865      jmp
2740          866      jmp
2741          867      jmp
2742          868      jmp
2743          869      jmp
2744          870      jmp
2745          871      jmp
2746          872      jmp
2747          873      jmp
2748          874      jmp
2749          875      jmp
2750          876      jmp
2751          877      jmp
2752          878      jmp
2753          879      jmp
2754          880      jmp
2755          881      jmp
2756          882      jmp
2757          883      jmp
2758          884      jmp
2759          885      jmp
2760          886      jmp
2761          887      jmp
2762          888      jmp
2763          889      jmp
2764          890      jmp
2765          891      jmp
2766          892      jmp
2767          893      jmp
2768          894      jmp
2769          895      jmp
2770          896      jmp
2771          897      jmp
2772          898      jmp
2773          899      jmp
2774          900      jmp
2775          901      jmp
2776          902      jmp
2777          903      jmp
2778          904      jmp
2779          905      jmp
2780          906      jmp
2781          907      jmp
2782          908      jmp
2783          909      jmp
2784          910      jmp
2785          911      jmp
2786          912      jmp
2787          913      jmp
2788          914      jmp
2789          915      jmp
2790          916      jmp
2791          917      jmp
2792          918      jmp
2793          919      jmp
2794          920      jmp
2795          921      jmp
2796          922      jmp
2797          923      jmp
2798          924      jmp
2799          925      jmp
2800          926      jmp
2801          927      jmp
2802          928      jmp
2803          929      jmp
2804          930      jmp
2805          931      jmp
2806          932      jmp
2807          933      jmp
2808          934      jmp
2809          935      jmp
2810          936      jmp
2811          937      jmp
2812          938      jmp
2813          939      jmp
2814          940      jmp
2815          941      jmp
2816          942      jmp
2817          943      jmp
2818          944      jmp
2819          945      jmp
2820          946      jmp
2821          947      jmp
2822          948      jmp
2823          949      jmp
2824          950      jmp
2825          951      jmp
2826          952      jmp
2827          953      jmp
2828          954      jmp
2829          955      jmp
2830          956      jmp
2831          957      jmp
2832          958      jmp
2833          959      jmp
2834          960      jmp
2835          961      jmp
2836          962      jmp
2837          963      jmp
2838          964      jmp
2839          965      jmp
2840          966      jmp
2841          967      jmp
2842          968      jmp
2843          969      jmp
2844          970      jmp
2845          971      jmp
2846          972      jmp
2847          973      jmp
2848          974      jmp
2849          975      jmp
2850          976      jmp
2851          977      jmp
2852          978      jmp
2853          979      jmp
2854          980      jmp
2855          981      jmp
2856          982      jmp
2857          983      jmp
2858          984      jmp
2859          985      jmp
2860          986      jmp
2861          987      jmp
2862          988      jmp
2863          989      jmp
2864          990      jmp
2865          991      jmp
2866          992      jmp
2867          993      jmp
2868          994      jmp
2869          995      jmp
2870          996      jmp
2871          997      jmp
2872          998      jmp
2873          999      jmp
2874          1000      jmp
2875          1001      jmp
2876          1002      jmp
2877          1003      jmp
2878          1004      jmp
2879          1005      jmp
2880          1006      jmp
2881          1007      jmp
2882          1008      jmp
2883          1009      jmp
2884          1010      jmp
2885          1011      jmp
2886          1012      jmp
2887          1013      jmp
2888          1014      jmp
2889          1015      jmp
2890          1016      jmp
2891          1017      jmp
2892          1018      jmp
2893          1019      jmp
2894          1020      jmp
2895          1021      jmp
2896          1022      jmp
2897          1023      jmp
2898          1024      jmp
2899          1025      jmp
2900          1026      jmp
2901          1027      jmp
2902          1028      jmp
2903          1029      jmp
2904          1030      jmp
2905          1031      jmp
2906          1032      jmp
2907          1033      jmp
2908          1034      jmp
2909          1035      jmp
2910          1036      jmp
2911          1037      jmp
2912          1038      jmp
2913          1039      jmp
2914          1040      jmp
2915          1041      jmp
2916          1042      jmp
2917          1043      jmp
2918          1044      jmp
2919          1045      jmp
2920          1046      jmp
2921          1047      jmp
2922          1048      jmp
2923          1049      jmp
2924          1050      jmp
2925          1051      jmp
2926          1052      jmp
2927          1053      jmp
2928          1054      jmp
2929          1055      jmp
2930          1056      jmp
2931          1057      jmp
2932          1058      jmp
2933          1059      jmp
2934          1060      jmp
2935          1061      jmp
2936          1062      jmp
2937          1063      jmp
2938          1064      jmp
2939          1065      jmp
2940          1066      jmp
2941          1067      jmp
2942          1068      jmp
2943          1069      jmp
2944          1070      jmp
2945          1071      jmp
2946          1072      jmp
2947          1073      jmp
2948          1074      jmp
2949          1075      jmp
2950          1076      jmp
2951          1077      jmp
2952          1078      jmp
2953          1079      jmp
2954          1080      jmp
2955          1081      jmp
2956          1082      jmp
2957          1083      jmp
2958          1084      jmp
2959          1085      jmp
2960          1086      jmp
2961          1087      jmp
2962          1088      jmp
2963          1089      jmp
2964          1090      jmp
2965          1091      jmp
2966          1092      jmp
2967          1093      jmp
2968          1094      jmp
2969          1095      jmp
2970          1096      jmp
2971          1097      jmp
2972          1098      jmp
2973          1099      jmp
2974          1100      jmp
2975          1101      jmp
2976          1102      jmp
2977          1103      jmp
2978          1104      jmp
2979          1105      jmp
2980          1106      jmp
2981          1107      jmp
2982          1108      jmp
2983          1109      jmp
2984          1110      jmp
2985          1111      jmp
2986          1112      jmp
2987          1113      jmp
2988          1114      jmp
2989          1115      jmp
2990          1116      jmp
2991          1117      jmp
2992          1118      jmp
2993          1119      jmp
2994          1120      jmp
2995          1121      jmp
2996          1122      jmp
2997          1123      jmp
2998          1124      jmp
2999          1125      jmp
3000          1126      jmp
3001          1127      jmp
3002          1128      jmp
3003          1129      jmp
3004          1130      jmp
3005          1131      jmp
3006          1132      jmp
3007          1133      jmp
3008          1134      jmp
3009          1135      jmp
3010          1136      jmp
3011          1137      jmp
3012          1138      jmp
3013          1139      jmp
3014          1140      jmp
3015          1141      jmp
3016          1142      jmp
3017          1143      jmp
3018          1144      jmp
3019          1145      jmp
3020          1146      jmp
3021          1147      jmp
3022          1148      jmp
3023          1149      jmp
3024          1150      jmp
3025          1151      jmp
3026          1152      jmp
3027          1153      jmp
3028          1154      jmp
3029          1155      jmp
3030          1156      jmp
3031          1157      jmp
3032          1158      jmp
3033          1159      jmp
3034          1160      jmp
3035          1161      jmp
3036          1162      jmp
3037          1163      jmp
3038          1164      jmp
3039          1165      jmp
3040          1166      jmp
3041          1167      jmp
3042          1168      jmp
3043          1169      jmp
3044          1170      jmp
3045          1171      jmp
3046          1172      jmp
3047          1173      jmp
3048          1174      jmp
3049          1175      jmp
3050          1176      jmp
3051          1177      jmp
3052          1178      jmp
3053          1179      jmp
3054          1180      jmp
3055          1181      jmp
3056          1182      jmp
3057          1183      jmp
3058          1184      jmp
3059          1185      jmp
3060          1186      jmp
3061          1187      jmp
3062          1188      jmp
3063          1189      jmp
3064          1190      jmp
3065          1191      jmp
3066          1192      jmp
3067          1193      jmp
3068          1194      jmp
3069          1195      jmp
3070          1196      jmp
3071          1197      jmp
3072          1198      jmp
3073          1199      jmp
3074          1200      jmp
3075          1201      jmp
3076          1202      jmp
3077          1203      jmp
3078          1204      jmp
3079          1205      jmp
3080          1206      jmp
3081          1207      jmp
3082          1208      jmp
3083          1209      jmp
3084          1210      jmp
3085          1211      jmp
3086          1212      jmp
3087          1213      jmp
3088          1214      jmp
3089          1215      jmp
3090          1216      jmp
3091          1217      jmp
3092          1218      jmp
3093          1219      jmp
3094          1220      jmp
3095          1221      jmp
3096          1222      jmp
3097          1223      jmp
3098          1224      jmp
30
```


2230 EFC0D3
2233 326D06
2236 71F86D
2239 EF4401
223C 346D03
223F 71F76D
2242 F3
2243 F0

```

286      call      swt1_expired
287 chk_sw2:
288      jbc      ios1_bak,2,chk_sw23
289      andb     ios1_bak,#11111011B      ; Clear bit 2
290      call      swt2_expired
291 chk_sw3:
292      jbc      ios1_bak,4,swt_int_done
293      andb     ios1_bak,#11110111B      ; Clear bit 3
294      call      swt3_expired
295
296 swt_int_done:
297      popf
298      ret      ; END OF SOFTWARE TIMER INTERRUPT ROUTINE
299
300 $eject
301
302
303 .....
304 .....
305 .....
306 .....
307 CSEG AT 2280H
308
309 hso_exec_int:
310
311      PUSHF
312      ldb      HSO_COMMAND,#30H
313      add      HSO_TIME,TIMER1,HSO0_dly
314
315      orb      port1,#00100000B      ; set P1.5
316      ld      Timer_2,TIMER2
317      jbs      Port1,1,in_mode2
318
319 in_mode1:
320      sub      tmp1,Timer_2,old_t2      ; Check count difference in tmp1
321      cmp      tmp1,#2
322      jh      end_sw20
323      set_mode0:
324      jbc      Port1,0,end_sw20      ; if already in mode 0
325      andb     Port1,#11111100B      ; Clear P1.0, P1.1 (set mode 0)
326      ldb      IOCO,#01010101B      ; enable all HSI
327      ldb      last_stat,zero
328      br      end_sw20
329
330 in_mode2:
331      sub      delta_p,timer_2,tmr2_old      ; get timer2 count difference
332      ld      tmr2_old,timer_2
333
334      jbc      direct,0,in_rev
335

```

530061-V3

270061-A2

intel

AP-248

10711

```

22B3 643C30      336  in_fwd: add    position,delta_p
22B6 A40032      337      addc   position+2,zero
22B9 2006        338      br     chk_mode
22BB 683C30      339
22BE A80032      340  in_rev: sub    position,delta_p
22C1            341      subc   position+2,zero
22C1            342
22C1 4866285C    343  chk_mode:
22C1 8905005C    344      sub     tmp1,Timer_2,old_t2      ; Check count difference in tmp1
22C9 D21C        345      cmp     tmp1,#5                  ; set model if count is too low
22C9            346      jgt     end_swt0              ; count <= 5
22CB 71FD0F      347
22CB 91010F      348  set_model:
22CE 810515      349      andb    Port1,#11111101B        ; Clear P1.1, set P1.0 (set mode 1)
22D1 A00400      350      orb     Port1,#00000001B
22D4 3F6DF4      351      ldb     IOCO,#00000101B          ; enable HSI 0 and 1
22D7 48B40A56    352      ld      zero,HSI_TIME
22E7            353      sub     last1_time,Timer1,min_hsil
22E7            354      ; set up so (time-last2_time)>min_hsil on next HSI
22E7            355  $EJECT
22DB A00400      356
22DB 717F6D      357  clr_hsi:
22DE 90166D      358      ld      ZERO,HSI_TIME
22E1 3F6DF4      359      andb    ios1_bak,#01111111B        ; clear bit 7
22E1            360      orb     ios1_bak,ios1
22E4            361      jbs     ios1_bak,7,clr_hsi      ; If hsi is triggered then clear hsi
22E7            362
22E7            363  end_swt0:
22E7 A02866      364      ld      old_t2,TIMER_2
22EA 71DF0F      365      andb    port1,#11011111B          ; clear P1.5
22ED F3          366      POPF
22EE F0          367      ret
22E7            368
22E7            369
22E7            370
22E7            371
22E7            372  SOFTWARE TIMER ROUTINE 2
22E7            373
22E7            374
22E7            375  CSEG AT 2380H
2380            376
2380            377  swt2_expired:
2380 F2          378      pushf
2381 B13A06      379      ldb     hso_command,#3AH          ; set swt_2
2384 44800A04    380      add     hso_time,timer1,swt2_dly
2381            381
2388 91040F      382      orb     port1,#00000100B          ; set port 1.2
238B 89FF075E    383      cmp     out_ptr,#7FFH
238F D104        384      bnh     pulsing
2391 A1F0015E    385      ld      out_ptr,#1FOH

```

270061-A3

```

2395 0605          386
2395 306E0C        387 pulsing: ld      412E 0100          ; 412E 0100 IN 0000
2395 306E0C        388 jbc      tr_col, 0, swt2_done
2395 306E0C        389 jpe      192E 0100          ; 192E 0100 IN 0000
239B C25F32        390 st      position+2, [out_ptr]+ ; position high, position low
239B C25F30        391 st      position, [out_ptr]+
239E C25F68        392
23A1 C25F6C        393 st      direct, [out_ptr]+
23A1 C25F6C        394 st      pwm_pwr, [out_ptr]+ ; 412E 0100 IN 0000
23A1 C25F6C        395 cwb     192E 0100          ; 192E 0100 IN 0000
23A1 C25F6C        396 jpe      192E 0100          ; store 8 bytes externally
23A1 C25F6C        397 jpe      192E 0100          ; store 8 bytes externally
23A4 48560A5C      398 swt2_done:
23A4 48560A5C      399 sub     tmp1, timer1, last1_time
23A8 8900185C      400 cmp     tmp1, #1800H
23AC D104          401 jnh     swt2_ret          ; keep (Timer1-last1_time)<2000H
23AE 65001056      402 cwb     192E 0100          ; 192E 0100 IN 0000
23B2 71FB0F        403 add     last1_time, #1000H
23B2 71FB0F        404 swt2_ret:
23B5 F3           405 andb   port1, #11111011B ; clear port1.2
23B5 F3           406 popf
23B6 F0           407 ret
23E7 810212        408 jpe      10C0 800000010B ; 10C0 800000010B
23E7 810212        409 $EJECT
23E7 810212        410
23E7 810212        411 HSI DATA AVAILABLE INTERRUPT ROUTINE
23E7 810212        412
23E7 810212        413
23E7 810212        414 ; This routine keeps track of the current time and position of the motor.
23E7 810212        415 ; The upper word of information is provided by the timer overflow routine.
23E7 810212        416
23E7 810212        417 CSEG AT 2400H
23E7 810212        418 now_mode_1: br      in_mode_1 ; used to save execution time for
23E7 810212        419 no_int1: cwb     br      no_int ; worst case loop
23E7 810212        420
23E7 810212        421 hsi_data_int: pushf
23E7 810212        422 orb     port1, #01000000B ; set P1.6
23E7 810212        423 andb   ios1_bak, #01111111B ; Clear ios1_bak.7
23E7 810212        424 orb     ios1_bak, ios1
23E7 810212        425 jbc     ios1_bak.7, no_int1 ; If hsi is not triggered then
23E7 810212        426 ; jump to no_int
23E7 810212        427 get_values:
23E7 810212        428 ld      timer_2, TIMER2
23E7 810212        429 andb   hsi_s0, HSI_STATUS, #01010101B
23E7 810212        430 ld      time, HSI_TIME
23E7 810212        431
23E7 810212        432 jbs     port1.0, now_mode_1 ; jump if in mode 1
23E7 810212        433
23E7 810212        434 In_mode_0:
23E7 810212        435 jbs     hsi_s0.0, a_rise
23E7 810212        436
23E7 810212        437

```

270061-A4

```

2421 3A6A2C      436      jbs      hsi_s0,2,a_fall
2424 3C6A4D      437      jbs      hsi_s0,4,b_rise
2427 3E6A5A      438      jbs      hsi_s0,6,b_fall
242A 2094        439      br       no_cnt
242C A0565B      440
242F A04056      441      a_rise: ld      last2_time,last1_time
2432 685840      442      ld       last1_time,time
2435 88B240      443      sub      time,last2_time
2438 D906        444      cmp      time,min_hsi
243A            445      jh       tst_statr
243C            446      ;set model-
243A 91010F      447      orb      Port1,#00000001B      ; Set P1 0 (in mode 1)
243D B10515      448      ldb      IOCO,#00000101B      ; Enable HSI 0 and 1
2440            449      tst_statr:
2440 3C6B5B      450      jbs      last_stat,6,going_fwd
2443 3C6B67      451      jbs      last_stat,4,going_rev
2446 3A6B50      452      jbs      last_stat,2,change_dir
2449 98006B      453      cmpb     last_stat,zero
244C DF46        454      je       first_time      ; first time in mode0
244E 27B2        455      br       no_intl
244F            456
2450 A0565B      457      a_fall: ld      last2_time,last1_time
2453 A04056      458      ld       last1_time,time
2456 685840      459      sub      time,last2_time
2459 88B240      460      cmp      time,min_hsi
245C D906        461      jh       tst_statf
245E 91010F      462      ;set model-
2461 B10515      463      orb      Port1,#00000001B      ; Set P1 0 (in mode 1)
2464            464      ldb      IOCO,#00000101B      ; Enable HSI 0 and 1
2464            465      $EJECT
2464            466      tst_statf:
2464 3C6B37      467      jbs      last_stat,4,going_fwd
2467 3E6B43      468      jbs      last_stat,6,going_rev
246A 386B2C      469      jbs      last_stat,0,change_dir
246D 98006B      470      cmpb     last_stat,zero
2470 DF22        471      je       first_time      ; first time in mode0
2472 2057        472      bne      no_intl
2473            473
2474 386B27      474      b_rise: jbs      last_stat,0,going_fwd
2477 3A6B33      475      jbs      last_stat,2,going_rev
247A 3E6B1C      476      jbs      last_stat,6,change_dir
247D 98006B      477      cmpb     last_stat,zero
2480 DF12        478      je       first_time      ; first time in mode0
2482 2047        479      br       no_intl
2483            480
2484 3A6B17      481      b_fall: jbs      last_stat,2,going_fwd
2487 386B23      482      jbs      last_stat,0,going_rev
248A 3C6B0C      483      jbs      last_stat,4,change_dir
248D 98006B      484      cmpb     last_stat,zero
2490 DF02        485      je       first_time      ; first time in mode0

```



```

24E3 D914          536      jh      check_max_time
24E4 88842C        537      cwb
24E5 4838402C      538      set_mode_2:
24E5 91020F        539      orb      Port1, #00000010B      ; Set P1 1 (in mode 2)
24E8 B10015        540      ldb      IOCO, #00000000B      ; Disable all HSI
24E8 A00400        541      mt_hsi: ld      zero, hsi_time      ; empty the hsi fifo
24EE 717F6D        542      andb     iosl_bak, #01111111B      ; clear the clear bit 7
24F1 90166D        543      cwb      iosl_bak, iosl      ;
24F4 3F6DF4        544      jbs      iosl_bak, 7, mt_hsi      ; If hsi is triggered then clear hsi
24F7 2012          545      br      done_chk
24F9              546
24F9 4858405C      547      check_max_time:
24FD 88865C        548      sub      tmp1, time, last2_time
24FD 88865C        549      cmp      tmp1, max_hsi1      ; max_hsi = addition to min_hsi for
2500 D109          550      ; total time
2500 D109          551      jnh      done_chk
2502              552
2502 71FC0F        553      set_mode_0:
2505 B15515        554      andb     Port1, #11111100B      ; clear P1.0.1 set mode 00
2508 B0Q06B        555      ldb      IOCO, #01010101B      ; Enable all HSI
2508 B0Q06B        556      ldb      last_stat, zero
2508 B0Q06B        557
2508 482C283C      558      done_chk:
250F 306808        559      sub      delta_p, timer_2, tmr2_old      ; get timer2 count difference
2512              560      jbc      direct, 0, add_rev
2512 643C30        561      add_fwd:
2515 A40032        562      add      position, delta_p
2518 27A3          563      addc     position+2, zero
251A 683C30        564      br      load_last
251A 683C30        565      add_rev:
251D A80032        566      sub      position, delta_p
2520 279B          567      subc     position+2, zero
2520 279B          568      br      load_last
2520 279B          569
2520 279B          570      $reject
2520 279B          571
2520 279B          572      ..... SOFTWARE TIMER ROUTINE 1 .....
2520 279B          573
2520 279B          574      CSEG: AT 2600H
2520 279B          575
2520 279B          576
2520 279B          577      swti_expired:
2520 279B          578
2520 279B          579      pushf
2520 279B          580      orb      port1, #10000000B      ; set port1 7
2520 279B          581
2520 279B          582      ldb      int_mask, #00001101B      ; enable HSI, Tof, HSO
2520 279B          583
2520 279B          584      ldb      HSO_COMMAND, #39H
2520 279B          585      add      HSO_TIME, TIMER1, swti_dly
2520 279B          586
2520 279B          587
2520 279B          588
2520 279B          589
2520 279B          590
2520 279B          591
2520 279B          592
2520 279B          593
2520 279B          594
2520 279B          595
2520 279B          596
2520 279B          597
2520 279B          598
2520 279B          599
2520 279B          600
2520 279B          601
2520 279B          602
2520 279B          603
2520 279B          604
2520 279B          605
2520 279B          606
2520 279B          607
2520 279B          608
2520 279B          609
2520 279B          610
2520 279B          611
2520 279B          612
2520 279B          613
2520 279B          614
2520 279B          615
2520 279B          616
2520 279B          617
2520 279B          618
2520 279B          619
2520 279B          620
2520 279B          621
2520 279B          622
2520 279B          623
2520 279B          624
2520 279B          625
2520 279B          626
2520 279B          627
2520 279B          628
2520 279B          629
2520 279B          630
2520 279B          631
2520 279B          632
2520 279B          633
2520 279B          634
2520 279B          635
2520 279B          636
2520 279B          637
2520 279B          638
2520 279B          639
2520 279B          640
2520 279B          641
2520 279B          642
2520 279B          643
2520 279B          644
2520 279B          645
2520 279B          646
2520 279B          647
2520 279B          648
2520 279B          649
2520 279B          650
2520 279B          651
2520 279B          652
2520 279B          653
2520 279B          654
2520 279B          655
2520 279B          656
2520 279B          657
2520 279B          658
2520 279B          659
2520 279B          660
2520 279B          661
2520 279B          662
2520 279B          663
2520 279B          664
2520 279B          665
2520 279B          666
2520 279B          667
2520 279B          668
2520 279B          669
2520 279B          670
2520 279B          671
2520 279B          672
2520 279B          673
2520 279B          674
2520 279B          675
2520 279B          676
2520 279B          677
2520 279B          678
2520 279B          679
2520 279B          680
2520 279B          681
2520 279B          682
2520 279B          683
2520 279B          684
2520 279B          685
2520 279B          686
2520 279B          687
2520 279B          688
2520 279B          689
2520 279B          690
2520 279B          691
2520 279B          692
2520 279B          693
2520 279B          694
2520 279B          695
2520 279B          696
2520 279B          697
2520 279B          698
2520 279B          699
2520 279B          700
2520 279B          701
2520 279B          702
2520 279B          703
2520 279B          704
2520 279B          705
2520 279B          706
2520 279B          707
2520 279B          708
2520 279B          709
2520 279B          710
2520 279B          711
2520 279B          712
2520 279B          713
2520 279B          714
2520 279B          715
2520 279B          716
2520 279B          717
2520 279B          718
2520 279B          719
2520 279B          720
2520 279B          721
2520 279B          722
2520 279B          723
2520 279B          724
2520 279B          725
2520 279B          726
2520 279B          727
2520 279B          728
2520 279B          729
2520 279B          730
2520 279B          731
2520 279B          732
2520 279B          733
2520 279B          734
2520 279B          735
2520 279B          736
2520 279B          737
2520 279B          738
2520 279B          739
2520 279B          740
2520 279B          741
2520 279B          742
2520 279B          743
2520 279B          744
2520 279B          745
2520 279B          746
2520 279B          747
2520 279B          748
2520 279B          749
2520 279B          750
2520 279B          751
2520 279B          752
2520 279B          753
2520 279B          754
2520 279B          755
2520 279B          756
2520 279B          757
2520 279B          758
2520 279B          759
2520 279B          760
2520 279B          761
2520 279B          762
2520 279B          763
2520 279B          764
2520 279B          765
2520 279B          766
2520 279B          767
2520 279B          768
2520 279B          769
2520 279B          770
2520 279B          771
2520 279B          772
2520 279B          773
2520 279B          774
2520 279B          775
2520 279B          776
2520 279B          777
2520 279B          778
2520 279B          779
2520 279B          780
2520 279B          781
2520 279B          782
2520 279B          783
2520 279B          784
2520 279B          785
2520 279B          786
2520 279B          787
2520 279B          788
2520 279B          789
2520 279B          790
2520 279B          791
2520 279B          792
2520 279B          793
2520 279B          794
2520 279B          795
2520 279B          796
2520 279B          797
2520 279B          798
2520 279B          799
2520 279B          800
2520 279B          801
2520 279B          802
2520 279B          803
2520 279B          804
2520 279B          805
2520 279B          806
2520 279B          807
2520 279B          808
2520 279B          809
2520 279B          810
2520 279B          811
2520 279B          812
2520 279B          813
2520 279B          814
2520 279B          815
2520 279B          816
2520 279B          817
2520 279B          818
2520 279B          819
2520 279B          820
2520 279B          821
2520 279B          822
2520 279B          823
2520 279B          824
2520 279B          825
2520 279B          826
2520 279B          827
2520 279B          828
2520 279B          829
2520 279B          830
2520 279B          831
2520 279B          832
2520 279B          833
2520 279B          834
2520 279B          835
2520 279B          836
2520 279B          837
2520 279B          838
2520 279B          839
2520 279B          840
2520 279B          841
2520 279B          842
2520 279B          843
2520 279B          844
2520 279B          845
2520 279B          846
2520 279B          847
2520 279B          848
2520 279B          849
2520 279B          850
2520 279B          851
2520 279B          852
2520 279B          853
2520 279B          854
2520 279B          855
2520 279B          856
2520 279B          857
2520 279B          858
2520 279B          859
2520 279B          860
2520 279B          861
2520 279B          862
2520 279B          863
2520 279B          864
2520 279B          865
2520 279B          866
2520 279B          867
2520 279B          868
2520 279B          869
2520 279B          870
2520 279B          871
2520 279B          872
2520 279B          873
2520 279B          874
2520 279B          875
2520 279B          876
2520 279B          877
2520 279B          878
2520 279B          879
2520 279B          880
2520 279B          881
2520 279B          882
2520 279B          883
2520 279B          884
2520 279B          885
2520 279B          886
2520 279B          887
2520 279B          888
2520 279B          889
2520 279B          890
2520 279B          891
2520 279B          892
2520 279B          893
2520 279B          894
2520 279B          895
2520 279B          896
2520 279B          897
2520 279B          898
2520 279B          899
2520 279B          900
2520 279B          901
2520 279B          902
2520 279B          903
2520 279B          904
2520 279B          905
2520 279B          906
2520 279B          907
2520 279B          908
2520 279B          909
2520 279B          910
2520 279B          911
2520 279B          912
2520 279B          913
2520 279B          914
2520 279B          915
2520 279B          916
2520 279B          917
2520 279B          918
2520 279B          919
2520 279B          920
2520 279B          921
2520 279B          922
2520 279B          923
2520 279B          924
2520 279B          925
2520 279B          926
2520 279B          927
2520 279B          928
2520 279B          929
2520 279B          930
2520 279B          931
2520 279B          932
2520 279B          933
2520 279B          934
2520 279B          935
2520 279B          936
2520 279B          937
2520 279B          938
2520 279B          939
2520 279B          940
2520 279B          941
2520 279B          942
2520 279B          943
2520 279B          944
2520 279B          945
2520 279B          946
2520 279B          947
2520 279B          948
2520 279B          949
2520 279B          950
2520 279B          951
2520 279B          952
2520 279B          953
2520 279B          954
2520 279B          955
2520 279B          956
2520 279B          957
2520 279B          958
2520 279B          959
2520 279B          960
2520 279B          961
2520 279B          962
2520 279B          963
2520 279B          964
2520 279B          965
2520 279B          966
2520 279B          967
2520 279B          968
2520 279B          969
2520 279B          970
2520 279B          971
2520 279B          972
2520 279B          973
2520 279B          974
2520 279B          975
2520 279B          976
2520 279B          977
2520 279B          978
2520 279B          979
2520 279B          980
2520 279B          981
2520 279B          982
2520 279B          983
2520 279B          984
2520 279B          985
2520 279B          986
2520 279B          987
2520 279B          988
2520 279B          989
2520 279B          990
2520 279B          991
2520 279B          992
2520 279B          993
2520 279B          994
2520 279B          995
2520 279B          996
2520 279B          997
2520 279B          998
2520 279B          999
2520 279B          1000
2520 279B          1001
2520 279B          1002
2520 279B          1003
2520 279B          1004
2520 279B          1005
2520 279B          1006
2520 279B          1007
2520 279B          1008
2520 279B          1009
2520 279B          1010
2520 279B          1011
2520 279B          1012
2520 279B          1013
2520 279B          1014
2520 279B          1015
2520 279B          1016
2520 279B          1017
2520 279B          1018
2520 279B          1019
2520 279B          1020
2520 279B          1021
2520 279B          1022
2520 279B          1023
2520 279B          1024
2520 279B          1025
2520 279B          1026
2520 279B          1027
2520 279B          1028
2520 279B          1029
2520 279B          1030
2520 279B          1031
2520 279B          1032
2520 279B          1033
2520 279B          1034
2520 279B          1035
2520 279B          1036
2520 279B          1037
2520 279B          1038
2520 279B          1039
2520 279B          1040
2520 279B          1041
2520 279B          1042
2520 279B          1043
2520 279B          1044
2520 279B          1045
2520 279B          1046
2520 279B          1047
2520 279B          1048
2520 279B          1049
2520 279B          1050
2520 279B          1051
2520 279B          1052
2520 279B          1053
2520 279B          1054
2520 279B          1055
2520 279B          1056
2520 279B          1057
2520 279B          1058
2520 279B          1059
2520 279B          1060
2520 279B          1061
2520 279B          1062
2520 279B          1063
2520 279B          1064
2520 279B          1065
2520 279B          1066
2520 279B          1067
2520 279B          1068
2520 279B          1069
2520 279B          1070
2520 279B          1071
2520 279B          1072
2520 279B          1073
2520 279B          1074
2520 279B          1075
2520 279B          1076
2520 279B          1077
2520 279B          1078
2520 279B          1079
2520 279B          1080
2520 279B          1081
2520 279B          1082
2520 279B          1083
2520 279B          1084
2520 279B          1085
2520 279B          1086
2520 279B          1087
2520 279B          1088
2520 279B          1089
2520 279B          1090
2520 279B          1091
2520 279B          1092
2520 279B          1093
2520 279B          1094
2520 279B          1095
2520 279B          1096
2520 279B          1097
2520 279B          1098
2520 279B          1099
2520 279B          1100
2520 279B          1101
2520 279B          1102
2520 279B          1103
2520 279B          1104
2520 279B          1105
2520 279B          1106
2520 279B          1107
2520 279B          1108
2520 279B          1109
2520 279B          1110
2520 279B          1111
2520 279B          1112
2520 279B          1113
2520 279B          1114
2520 279B          1115
2520 279B          1116
2520 279B          1117
2520 279B          1118
2520 279B          1119
2520 279B          1120
2520 279B          1121
2520 279B          1122
2520 279B          1123
2520 279B          1124
2520 279B          1125
2520 279B          1126
2520 279B          1127
2520 279B          1128
2520 279B          1129
2520 279B          1130
2520 279B          1131
2520 279B          1132
2520 279B          1133
2520 279B          1134
2520 279B          1135
2520 279B          1136
2520 279B          1137
2520 279B          1138
2520 279B          1139
2520 279B          1140
2520 279B          1141
2520 279B          1142
2520 279B          1143
2520 279B          1144
2520 279B          1145
2520 279B          1146
2520 279B          1147
2520 279B          1148
2520 279B          1149
2520 279B          1150
2520 279B          1151
2520 279B          1152
2520 279B          1153
2520 279B          1154
2520 279B          1155
2520 279B          1156
2520 279B          1157
2520 279B          1158
2520 279B          1159
2520 279B          1160
2520 279B          1161
2520 279B          1162
2520 279B          1163
2520 279B          1164
2520 279B          1165
2520 279B          1166
2520 279B          1167
2520 279B          1168
2520 279B          1169
2520 279B          1170
2520 279B          1171
2520 279B          1172
2520 279B          1173
2520 279B          1174
2520 279B          1175
2520 279B          1176
2520 279B          1177
2520 279B          1178
2520 279B          1179
2520 279B          1180
2520 279B          1181
2520 279B          1182
2520 279B          1183
2520 279B          1184
2520 279B          1185
2520 279B          1186
2520 279B          1187
2520 279B          1188
2520 279B          1189
2520 279B          1190
2520 279B          1191
2520 279B          1192
2520 279B          1193
2520 279B          1194
2520 279B          1195
2520 279B          1196
2520 279B          1197
2520 279B          1198
2520 279B          1199
2520 279B          1200
2520 279B          1201
2520 279B          1202
2520 279B          1203
2520 279B          1204
2520 279B          1205
2520 279B          1206
2520 279B          1207
2520 279B          1208
2520 279B          1209
2520 279B          1210
2520 279B          1211
2520 279B          1212
2520 279B          1213
2520 279B          1214
2520 279B          1215
2520 279B          1216
2520 279B          1217
2520 279B          1218
2520 279B          1219
2520 279B          1220
2520 279B          1221
2520 279B          1222
2520 279B          1223
2520 279B          1224
2520 279B          1225
2520 279B          1226
2520 279B          1227
2520 279B          1228
2520 279B          1229
2520 279B          1230
2520 279B          1231
2520 279B          1232
2520 279B          1233
2520 279B          1234
2520 279B          1235
2520 279B          1236
2520 279B          1237
2520 279B          1238
2520 279B          1239
2520 279B          1240
2520 279B          1241
2520 279B          1242
2520 279B          1243
2520 279B          1244
2520 279B          1245
2520 279B          1246
2520 279B          1247
2520 279B          1248
2520 279B          1249
2520 279B          1250
2520 279B          1251
2520 279B          1252
2520 279B          1253
2520 279B          1254
2520 279B          1255
2520 279B          1256
2520 279B          1257
2520 279B          1258
2520 279B          1259
2520 279B          1260
2520 279B          1261
2520 279B          1262
2520 279B          1263
2520 279B          1264
2520 279B          1265
2520 279B          1266
2520 279B          1267
2520 279B          1268
2520 279B          1269
2520 279B          1270
2520 279B          1271
2520 279B          1272
2520 279B          1273
2520 279B          1274
2520 279B          1275
2520 279B          1276
2520 279B          1277
2520 279B          1278
2520 279B          1279
2520 279B          1280
2520 279B          1281
2520 279B          1282
2520 279B          1283
2520 279B          1284
2520 279B          1285
2520 279B          1286
2520 279B          1287
2520 279B          1288
2520 279B          1289
2520 279B          1290
2520 279B          1291
2520 279B          1292
2520 279B          1293
2520 279B          1294
2520 279B          1295
2520 279B          1296
2520 279B          1297
2520 279B          1298
2520 279B          1299
```

270061-A8

```

265B D9F1          634      jh      ld_max      , position_error>brake_point
265A              635
265A 880050        636      braking
265D D602          637      cmp      pos_delta,zero
265F 0350          638      jge      chk_delta
2661              639      neg      pos_delta
2661 887650        640      chk_delta:
2664 D10D          641      cmp      pos_delta,vel_pnt      ; velocity = pos_delta/sample_time
2664 D10D          642      jnh      hold_position      ; jmp if ABS(velocity) < vel_pnt
2666 B0726C        643
2669 B06824        644      brake:   ldb      pwm_pwr,max_brk
266C 1224          645      ldb      tmp,direct      ; If braking apply power in opposite
266E B02469        646      notb     tmp      ; direction of current motion
266E B02469        647      ldb      pwm_dir,tmp
2671 2030          648
2671 2030          649      br      ld_pwr
2671 2030          650
2673              651      Hold_position:      , position hold mode
2673 8902003B       652      cmp      pos_err,#02
2677 D906          653      jh      calc_out      ; if position error < 2 then turn off power
2679 0126          654      clr      tmp+2
267B 015A          655      clr      boost
267D 201F          656      BR      output
267D 201F          657
267F              658      calc_out:
267F 5DFF7424       659      mulub    tmp,max_hold,#255
2683 6C3B24        660      mulu     tmp,pos_err      ; Tmp = pos_err * max_hold
2686 880050        661      cmp      pos_delta,zero
2689 D709          662      jne      no_bst
268B 6504005A      663      add      boost,#04      ; Boost is integral control
268F 645A26        664      add      tmp+2,boost      ; TMP+2 = MSB(pos_err*max_hold)
2692 2002          665      br
2694 015A          666      no_bst:   clr      boost
2696 887426        667      ck_max:   cmp      tmp+2,max_hold
2699 D103          668      jnh      output
269B A07426        669      maxed:   ld      tmp+2,max_hold
269E B0266C        670      output:  ldb      pwm_pwr,tmp+2
269E B0266C        671
269E B0266C        672
26A1              673      chk_sanity:
26A1 2000          674      br      ld_pwr
26A1 2000          675
26A1 2000          676      ;;
26A1 2000          677      $EJECT    FI
26A1 2000          678
26A3              679      ld_pwr:
26A3 B06C64        680      ldb      rpwr,pwm_pwr
26A6 1264          681      notb     rpwr
26AB 38690A        682      jbs      pwm_dir,0,p2fwd
26AB 38690A        683
26AB 38690A        684

```



```

26AB FA          684 p2bkwd: DI
26AC 717F10      685         andb    port2,#01111111B      ; clear P2.7
26AF B06417      686         ldb     pwm_control,rpwr
26B2 FB          687         EI
26B3 2008        688         br      pwrset
26B5 FA          689 p2fwd: DI
26B6 918010      690         orb     port2,#10000000B      ; set P2.7
26B9 B06417      691         ldb     pwm_control,rpwr
26BC FB          692         EI
                693
26BD             694 pwrset:
26BD 88004A      695         cmp     time_err+2,zero      ; do pos_table when err is negative
26C0 D225        696         jgt     end_p
                697         br      end_p
                698
26C2 89202962    699         cmp     nxt_pos,#(32+pos_table)
26C6 DE06        700         jlt     end      ; jump if lower
26C8 A1002962    701         ld      nxt_pos,#pos_table
26CC 0142        702         clr     time+2
26CE 5A3C 8700E000 703         get_vals: qcm 0041H' 0060H      ; next time' base1
                704         qct 00000000H      ; position 1
                705         ld      qcm des_pos,[nxt_pos]+      ; next time' base1
                706         ld      qct des_pos+2,[nxt_pos]+    ; position 2
                707         ld      qcm des_time+2,[nxt_pos]+    ; next time' base1
                708         ld      qct max_pwr,[nxt_pos]+      ; position 2
                709         ld      qcm max_brk,max_pwr      ; next time' base1
                710         add     des_pos,offset      ; position 4
                711         addc    des_pos+2,zero
                712         sub     last_pos_err,des_pos,position 4 ; base1
                713         qct 0041E000H      ; position 2
                714         end_p: andb    port1,#01111111B      ; clear P1.7
                715         qct 00000000H      ; position 5
                716         popf    qcm 0040H' 0040H      ; next time' base1
                717         ret     qct 0000F000H      ; position 7
                718         qcm 0050H' 0080H      ; next time' base1
                719         $EJECT qct 00000000H      ; position 0
                720
                721         ; .....
                722         ; ..... main program .....
                723         ; .....
                724
                725
                726         CSEG at 2800H
                727
                728         MAIN_PROG: qcm 0041' 0041' 000001000B      ; control bit 0
                729         orb     ios1_bak,ios1
                730         jbc     ios1_bak.6,control
                731         andb    ios1_bak,#10111111B      ; clear ios1_bak.6
                732         xorb    port1,#00010000B      ; Compl Bit P1.4
                733         call    HSI_DATA_INT,00010101B ; prevent lockup
                734         qcm 0041E011'

```

270061-B0

```

280F          734 control:
5800 280F 912D08 133 735      orb     int_mask,#00101101B ; enable hsi, hso, swt, tovf interrupts
5801 2812 FD      135 736      nop
5802 2813 FD      137 737      nop
5803 2814 FD      139 738      nop
5804 2815 E06FFD 139 739      djnz    main_dly,$
5805 2818 FD      139 740      nop
5806 2819 95080F 139 741      xorb     port1,#00001000B ; compliment p1.3
5807 281C 27E2    139 742      BR      MAIN_PROG
                    139 743
                    139 744
2900          139 745      CSEG AT 2900H
                    139 746
2900          139 747      pos_table:
                    139 748
2900 00000000    139 749      dcl     00000000H ; position 0
2904 20008000    139 750      dcw     0020H, 0080H ; next time, power
59E0 2908 00C00000 139 751      dcl     0000C000H ; position 1
59E1 290C 40004000 139 752      dcw     0040H, 0040H ; next time, power
2910 00000000    139 753      dcl     00000000H ; position 2
59E1 2914 6000C000 139 754      dcw     0060H, 00C0H ; next time, power
2918 0080FFFF    139 755      dcl     0FFFF8000H ; position 3
59E2 291C 80008000 139 756      dcw     0080H, 0080H ; next time, power
59E3 2920 00080000 139 757      dcl     00000800H ; position 4
59D0 2924 58008000 139 758      dcw     0058H, 0080H ; next time, power
59D1 2928 00300000 139 759      dcl     00003000H ; position 5
59D2 292C 7000FF00 139 760      dcw     0070H, 00FFH ; next time, power
59D3 2930 00000000 139 761      dcl     00000000H ; position 6
59C0 2934 9000F000 139 762      dcw     0090H, 00F0H ; next time, power
2938 00000000    139 763      dcl     00000000H ; position 7
59C1 293C 9100F000 139 764      dcw     0091H, 00F0H ; next time, power
59C2 2940 0145    139 765      END
59C3 2944 0145    139 766      END
59C4 2948 0145    139 767      END
59C5 294C 0145    139 768      END
ASSEMBLY COMPLETED, NO ERROR(S) FOUND.
59C0 0553    139 769      END
59B0 08000F    139 770      END
59B1 08000F    139 771      END
59B2 08000F    139 772      END
59B3 08000F    139 773      END
59B4 08000F    139 774      END
59B5 08000F    139 775      END
59B6 08000F    139 776      END
59B7 08000F    139 777      END
59B8 08000F    139 778      END
59B9 08000F    139 779      END
59BA 08000F    139 780      END
59BB 08000F    139 781      END
59BC 08000F    139 782      END
59BD 08000F    139 783      END
59BE 08000F    139 784      END
59BF 08000F    139 785      END
59C0 08000F    139 786      END
59C1 08000F    139 787      END
59C2 08000F    139 788      END
59C3 08000F    139 789      END
59C4 08000F    139 790      END
59C5 08000F    139 791      END
59C6 08000F    139 792      END
59C7 08000F    139 793      END
59C8 08000F    139 794      END
59C9 08000F    139 795      END
59CA 08000F    139 796      END
59CB 08000F    139 797      END
59CC 08000F    139 798      END
59CD 08000F    139 799      END
59CE 08000F    139 800      END
59CF 08000F    139 801      END
59D0 08000F    139 802      END
59D1 08000F    139 803      END
59D2 08000F    139 804      END
59D3 08000F    139 805      END
59D4 08000F    139 806      END
59D5 08000F    139 807      END
59D6 08000F    139 808      END
59D7 08000F    139 809      END
59D8 08000F    139 810      END
59D9 08000F    139 811      END
59DA 08000F    139 812      END
59DB 08000F    139 813      END
59DC 08000F    139 814      END
59DD 08000F    139 815      END
59DE 08000F    139 816      END
59DF 08000F    139 817      END
59E0 08000F    139 818      END
59E1 08000F    139 819      END
59E2 08000F    139 820      END
59E3 08000F    139 821      END
59E4 08000F    139 822      END
59E5 08000F    139 823      END
59E6 08000F    139 824      END
59E7 08000F    139 825      END
59E8 08000F    139 826      END
59E9 08000F    139 827      END
59EA 08000F    139 828      END
59EB 08000F    139 829      END
59EC 08000F    139 830      END
59ED 08000F    139 831      END
59EE 08000F    139 832      END
59EF 08000F    139 833      END
59F0 08000F    139 834      END
59F1 08000F    139 835      END
59F2 08000F    139 836      END
59F3 08000F    139 837      END
59F4 08000F    139 838      END
59F5 08000F    139 839      END
59F6 08000F    139 840      END
59F7 08000F    139 841      END
59F8 08000F    139 842      END
59F9 08000F    139 843      END
59FA 08000F    139 844      END
59FB 08000F    139 845      END
59FC 08000F    139 846      END
59FD 08000F    139 847      END
59FE 08000F    139 848      END
59FF 08000F    139 849      END

```

270061-B1

of what the FFT does. There are four program modules discussed in this application note:

conversion and printing.
for console out and hexadecimal to decimal
connected printer. Routines are provided
PLOTSP - Plots the contents of a buffer to a serially

An FFT Algorithm For MCS®-96 Products Including Supporting Routines and Examples

IRA HOECHT
ECO APPLICATIONS

October 1988

applications, so while it is directly applicable to a specific set of applications, it is indirectly applicable to many more.

Supporting Examples

848 A.P. 148

SD PROGRAM OVERVIEW

IS ENGINEER

IRA HORDEN

ECO APPLICATIONS ENGINEER

1.0 INTRODUCTION

Intel's 8096 is a 16-bit microcontroller with processing power sufficient to perform many tasks which were previously done by microprocessors or special building block computers. A new field of applications is opened by having this much power available on a single chip controller.

The 8096 can be used to increase the performance of existing designs based on 8051s or similar 8-bit controllers. In addition, it can be used for Digital Signal Processing (DSP) applications, as well as matrix manipulations and other processing oriented tasks. One of the tasks that can be performed is the calculation of a Fast Fourier Transform (FFT). The algorithm used is similar to that in many DSP and matrix manipulation applications, so while it is directly applicable to a specific set of applications, it is indirectly applicable to many more.

FFTs are most often used in determining what frequencies are present in an analog signal. By providing a tool to identify specific waveforms by their frequency components, FFTs can be used to compare signals to one another or to set patterns. This type of procedure is used in speech detection and engine knock sensors. FFTs also have uses in vision systems where they identify objects by comparing their outlines, and in radar units to detect the dopler shift created by moving objects.

This application note discusses how FFTs can be calculated using Intel's MCS®-96 microcontrollers. A review of fourier analysis is presented, along with the specific code required for a 64 point real FFT. Throughout this application note, it is assumed that the reader has a working knowledge of the 8096. For those without this background the following two publications will be helpful:

1986 Microcontroller Handbook
Using the 8096, AP-248

These books are listed in the bibliography, along with other good sources of information on the MCS-96 product family and on Fast Fourier Transforms.

2.0 PROGRAM OVERVIEW

This application note contains program modules which are combined to create a program which performs an FFT on an analog signal sampled by the on-board ADC (Analog to Digital Converter) of the 8097. The results of the FFT are then provided over the serial

channel to a printer or terminal which displays the results. In the applications listed in the previous section, the data from this FFT program would be used directly by another program instead of being plotted. However, the plotted results are used here to provide an example of what the FFT does. There are four program modules discussed in this application note:

FFTRUN - Runs a 64 point FFT on its data buffer. It produces 32 14-bit complex output values and 32 14-bit output magnitudes. A fast square root routine and log conversion routine are included.

A2DCON - Fills one of two buffers with analog values at a set sample rate. The sample time can be as fast as 50 microseconds using 8x9xBH components.

PLOTSP - Plots the contents of a buffer to a serially connected printer. Routines are provided for console out and hexadecimal to decimal conversion and printing.

FTMAIN - The main module which controls the other modules.

Each of the modules will be described separately. In order to better understand how the programs work together, a brief tutorial on FFTs will be presented first, followed by descriptions of the programs in the order listed above.

The final program uses 64 real data points, taken from either a table or analog input 1. Each of the data points is a 16-bit signed number. The processing takes 12.5 milliseconds when internal RAM is used as the data space. If external RAM is used, 14 milliseconds are required. Larger FFTs can be performed by slightly modifying the programs. A 256-point FFT would take approximately 65 milliseconds, and a 1024-point version would require about 300 milliseconds.

In the program presented, the analog sampling time is set for 1 sample every 100 microseconds, providing the 64 samples in 6.4 milliseconds. The sampling time can be reduced to around 60 microseconds per point by changing a variable, and less than 50 microseconds by using the 8x9xBH series of parts, since they have a 22 microsecond A to D conversion time.

The programs are set up to be run in a sequence instead of concurrently. This provides the fastest operation if the sampling speed were reduced to the minimum possible. For the fastest operation above about 80 microseconds a sample, the programs could be run concurrently, but this would require some minor modifications of the program. Figure 1 shows the timing of the program as presented.

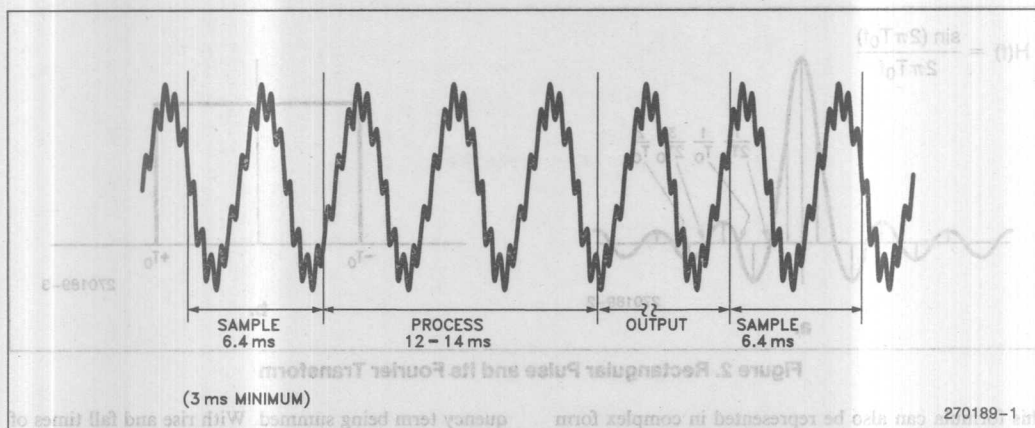


Figure 1. Timing of the FFT Program

These programs have run in the Intel Microcontroller Operation Application's Lab and produced the results presented in this application note. Since the programs have not undergone any further testing, we cannot guarantee them to be bug proof. We, therefore, recommend that they be thoroughly tested before being used for other than demonstration purposes.

3.0 FOURIER TRANSFORMS

A Fourier Transform is a useful analytical tool that is frequently ignored due to its mathematically oriented derivations. This is unfortunate, since Fourier transforms can be used without fully understanding the mathematics behind them. Of course, if one understands the theory behind these transforms, they become much more powerful.

The majority of this application note deals with how a Fast Fourier Transform (FFT) can be used for spectrum analysis. This procedure takes an input signal and separates it into its frequency components. One can almost treat the FFT as a black box, which has as its output, the frequency components and magnitudes of the input signal, much like a spectrum analyzer.

From a mathematical standpoint, Fourier Transforms change information in the time domain into the frequency domain. The theory behind the Fourier transform stems from Fourier analysis, also called frequency analysis.

There are many books on the topic of Fourier analysis, several of which are listed in the bibliography. In this application note, only the pertinent formulas and uses will be presented, not their derivations.

The main idea in Fourier analysis is that a function can be expressed as a summation of sinusoidal functions of different frequencies, phase angles, and magnitudes. This idea is represented by the Fourier Integral:

$$H(f) = \int_{-\infty}^{\infty} h(t) e^{-j2\pi ft} dt \quad (1)$$

Where: $H(f)$ is a function of frequency
 $h(t)$ is a function of time

Since

$$e^{-j\theta} = \cos \theta - j \sin \theta \quad (2)$$

$$H(f) = \int_{-\infty}^{\infty} h(t) (\cos (2\pi ft) - j \sin (2\pi ft)) dt \quad (3)$$

Figure 2 shows a rectangular pulse and its Fourier transform. Note that the results in the frequency domain are continuous rather than discrete. The horizontal axis in Figure 2a is frequency, while that of Figure 2b is time.

In a simplified case, the varying phase angles can be removed, and the integral changed to a summation, known as a Fourier Series. All periodic functions can be described in this way. This series, as shown below, can help provide a more graphical understanding of Fourier analysis.

$$y(t) = \frac{a_0}{2} + \sum_{n=1}^{\infty} [a_n \cos (2\pi n f_0 t) + b_n \sin (2\pi n f_0 t)] \quad (4)$$

for $n = 1$ to ∞

Where $f_0 = \frac{1}{T_0}$, the fundamental frequency.

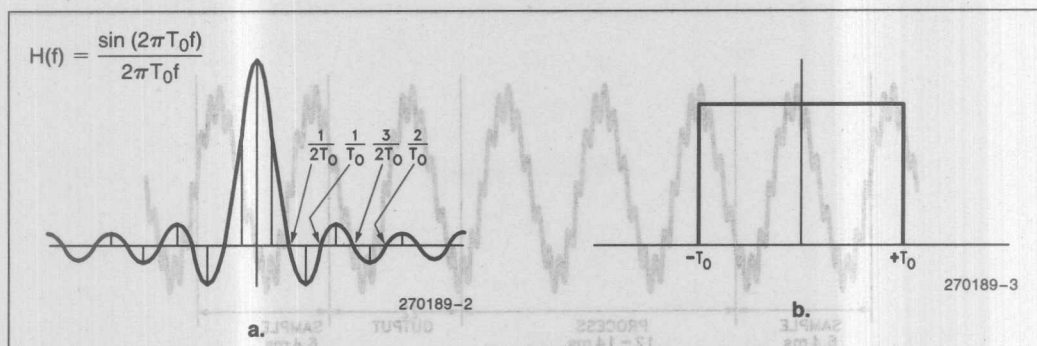


Figure 2. Rectangular Pulse and Its Fourier Transform

This formula can also be represented in complex form as:

$$\sum_{n=-\infty}^{\infty} a_n e^{j2\pi n f_0 t} \quad (5)$$

The Fourier series for a square wave is

$$\sum_{k=0}^{\infty} \frac{\sin((2k+1)2\pi f_0 t)}{(2k+1)} \quad (6)$$

If these sinusoids are summed, a square wave will be formed. Figure 3 shows the graphical summation of the first 3 terms of the series. Since the higher frequencies contribute to the squareness of the waveform at the corners, it is reasonable to compare only the flatness of the top of the waveform. The sharpness or risetime of the waveform can be determined by the highest fre-

quency term being summed. With rise and fall times of 10% of the period, the waveform generated by the first 3 terms is within 20% of ideal. At 7 terms it is within 10%, and at 20 terms it is within 5%. With a 5% risetime, it is within 20% of ideal after 5 terms, 10% after 13 terms and 5% after 32 terms. Figure 4 shows the resultant waveforms after the summation of 7, 15 and 30 terms.

Fourier analysis can be used on equation 4 to find the coefficients a_n and b_n . To make this process easier to use with a computer, a discrete form, rather than a continuous one, must be used. The discrete Fourier transform, shown in Equation 7, is a good approximation to the continuous version. The closeness of the approximation depends on several conditions which will be discussed later. The input to this transform is a set of N equally spaced samples of a waveform taken over a period of NT . The period NT is frequently referred to as the "Sampling Window"

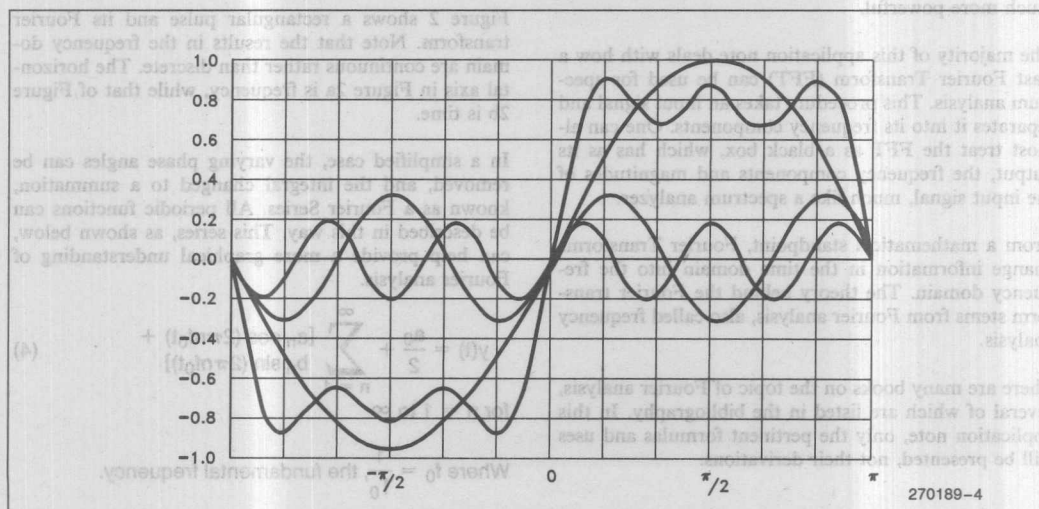


Figure 3. Graphical Summation of Sinewaves

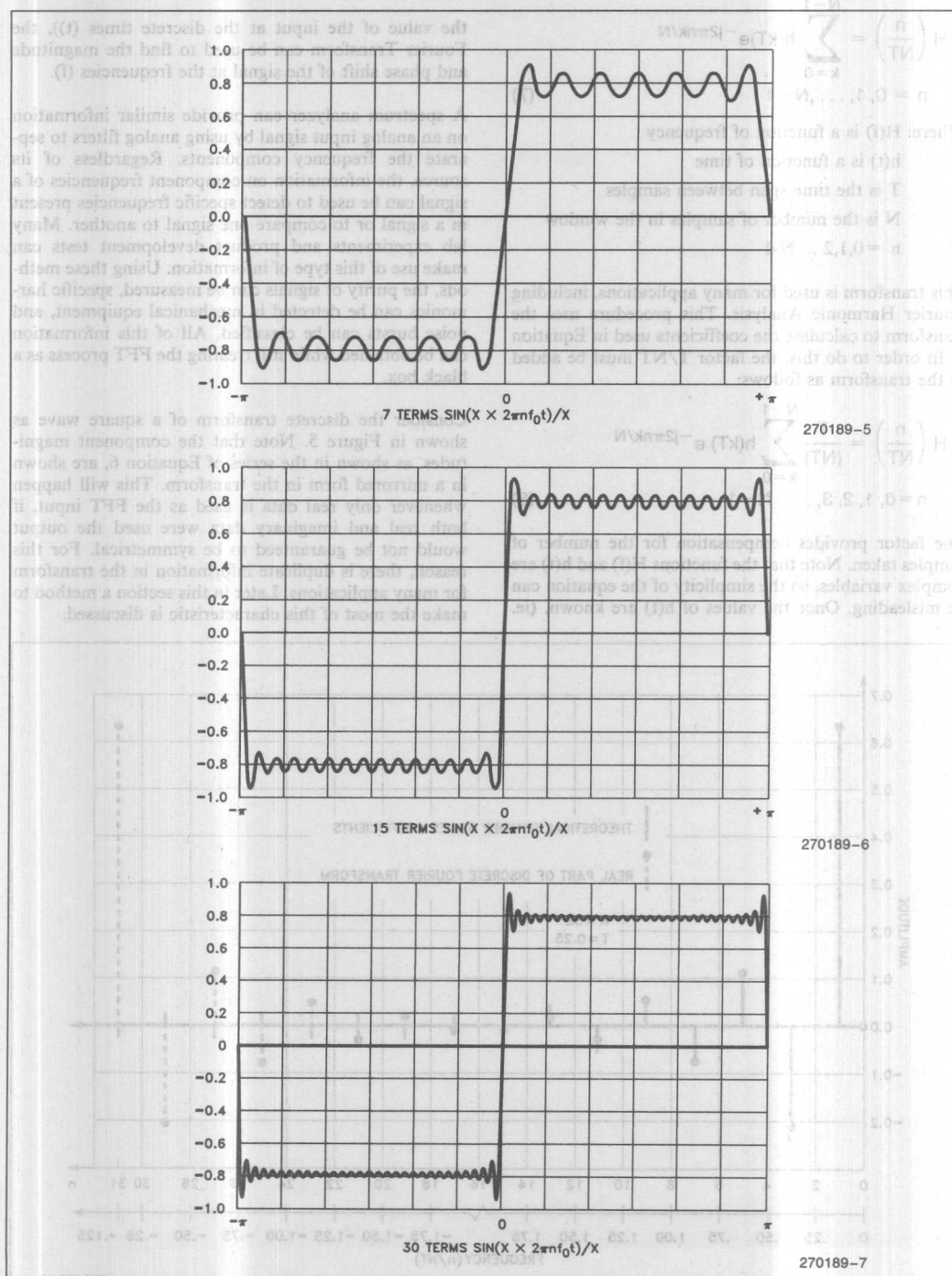


Figure 4. Square Wave from Sinusoids

$$H\left(\frac{n}{NT}\right) = \sum_{k=0}^{N-1} h(kT)e^{-j2\pi nk/N} \quad (7)$$

$n = 0, 1, \dots, N-1$

Where: $H(f)$ is a function of frequency

$h(t)$ is a function of time

T is the time span between samples

N is the number of samples in the window

$n = 0, 1, 2, \dots, N-1$

This transform is used for many applications, including Fourier Harmonic Analysis. This procedure uses the transform to calculate the coefficients used in Equation 5. In order to do this, the factor T/NT must be added to the transform as follows:

$$H\left(\frac{n}{NT}\right) = \frac{T}{NT} \sum_{k=0}^{N-1} h(kT)e^{-j2\pi nk/N} \quad (8)$$

$n = 0, 1, 2, 3, \dots, N-1$

The factor provides compensation for the number of samples taken. Note that the functions $H(f)$ and $h(t)$ are complex variables, so the simplicity of the equation can be misleading. Once the values of $h(t)$ are known, (ie.

the value of the input at the discrete times (t)), the Fourier Transform can be used to find the magnitude and phase shift of the signal at the frequencies (f).

A spectrum analyzer can provide similar information on an analog input signal by using analog filters to separate the frequency components. Regardless of its source, the information on component frequencies of a signal can be used to detect specific frequencies present in a signal or to compare one signal to another. Many lab experiments and product development tests can make use of this type of information. Using these methods, the purity of signals can be measured, specific harmonics can be detected in mechanical equipment, and noise bursts can be classified. All of this information can be obtained while still treating the FFT process as a black box.

Consider the discrete transform of a square wave as shown in Figure 5. Note that the component magnitudes, as shown in the series of Equation 6, are shown in a mirrored form in the transform. This will happen whenever only real data is used as the FFT input, if both real and imaginary data were used the output would not be guaranteed to be symmetrical. For this reason, there is duplicate information in the transform for many applications. Later in this section a method to make the most of this characteristic is discussed.

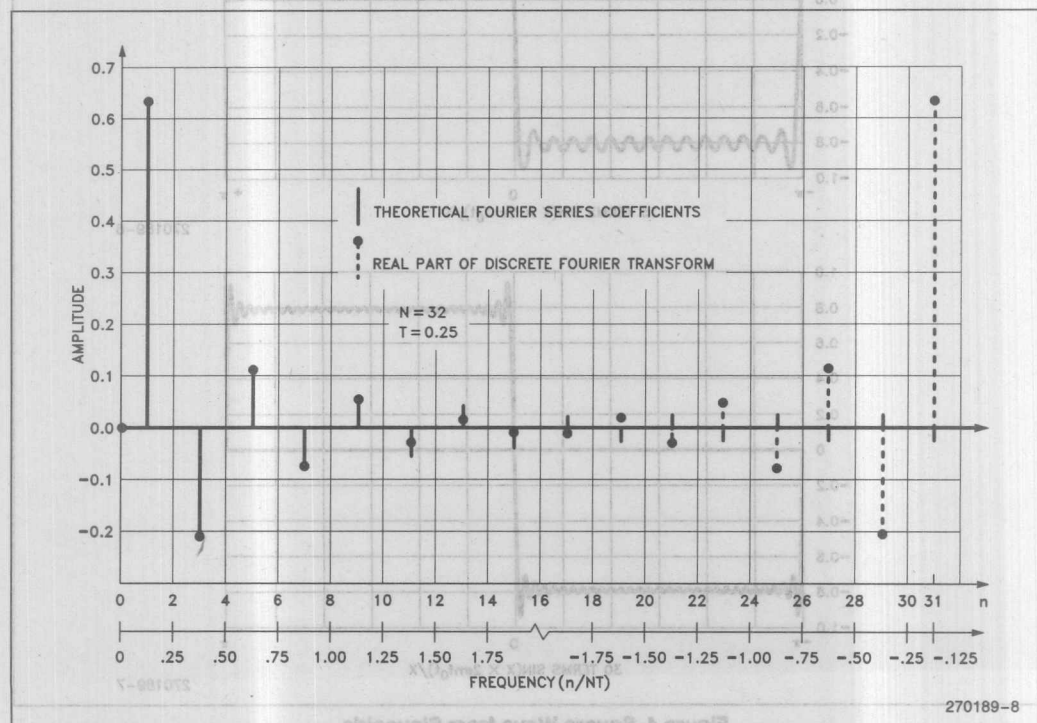


Figure 5. Discrete Transform of a Square Wave

If one looks at Equation 8, it can be seen that the calculation of a discrete Fourier transform requires N squared complex multiplications. If N is large, the calculation time can easily become unrealistic for real-time applications. For example, if a complex multiplication takes 40 microseconds, at $N = 16$, 10 milliseconds would be used for calculation, while at $N = 128$, over half a second would be needed. A Fast Fourier Transform is an algorithm which uses less multiplications, and is therefore faster. To calculate the actual time savings, it is first necessary to understand how a FFT works.

4.0 THE FFT ALGORITHM

The FFT algorithm makes use of the periodic nature of waveforms and some matrix algebra tricks to reduce the number of calculations needed for a transform. A more complete discussion of this is in Appendix A, however, the areas that need to be understood to follow the algorithm are presented here. This information need not be read if the reader's intent is to use the program and not to understand the mathematical process of the algorithm.

To simplify notation the following substitutions are made in Equation 8.

$$W = e^{-j2\pi/N}$$

$$k = kT$$

$$n = \frac{n}{NT}$$

The resultant equation being

$$x(n) = \sum_{k=0}^{N-1} n(k)W^{nk} \quad (9)$$

Expressed as a matrix operation

$$\begin{bmatrix} X(1) \\ X(2) \\ X(3) \\ \vdots \\ X(N-1) \end{bmatrix} = \begin{bmatrix} W^0 & W^0 & W^0 & \dots & W^0 \\ W^0 & W^1 & W^2 & \dots & W^N \\ W^0 & W^2 & W^4 & \dots & W^{2N} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ W^0 & W^{(N-1)} & W^{2(N-1)} & \dots & W^{(N-1)^2} \end{bmatrix} \begin{bmatrix} X_0(0) \\ X_0(1) \\ X_0(2) \\ \vdots \\ X_0(N-1) \end{bmatrix}$$

A brief review of matrix properties can be found in Appendix A. Because of the periodic nature of W the following is true:

$$W^{nk} \text{ MOD } N = W^{nk} \quad (10)$$

$$= \cos(2\pi nk/N) - j \sin(2\pi nk/N)$$

$$W^0 = 1 \text{ therefore, if } nk \text{ MOD } N = 0, W^{nk} = 1$$

This reduces the calculations as several of the W terms go to 1 and the highest power of W is N . All of W values are complex, so most of the operations will have to be complex operations. We will continue to use only the W , $X(n)$ and $X_0(k)$ symbols to represent these complex quantities.

The FFT algorithm we will use requires that N be an integral power of 2. Other FFT algorithms do not have this restriction, but they are more complex to understand and develop. Additionally, for the relatively small values of N we are using this restriction should not provide much of a problem. We will define EXPONENT as log base 2 of N . Therefore,

$$N = 2^{\text{EXPONENT}}$$

The magic of the FFT, (as detailed in Appendix A), involves factoring the matrix into EXPONENT matrices, each of which has all zeros except for a 1 and a W^{nk} term in each row. When these matrices are multiplied together the result is the same as that of the multiplication indicated in Equation 9, except that the rows are interchanged and there are fewer non-trivial multiplications. To reorder the rows, and thus make the information useful, it is necessary to perform a procedure called "Bit Reversal".

This process requires that N first be converted to a binary number. The least significant bit (lsb) is swapped with the most significant bit (msb). Then the next lsb is swapped with the next msb, and so on until all bits have been swapped once. For $N=8$, 3 bits are used, and the values for N and their bit reversals are shown below:

Number	Binary	Bit Reversal	Decimal BR
0	000	000	0
1	001	100	4
2	010	010	2
3	011	110	6
4	100	001	1
5	101	101	5
6	110	011	3
7	111	111	7

Recall that the FFT of real data provides a mirrored image output, but the FFT algorithm can accept inputs with both real and imaginary components. Since the inputs for harmonic analysis provided by a single A to D are real, the FFT algorithm is doing a lot of calculations with one input term equal to zero. This is obviously not very efficient. More information for a given size transform can be obtained by using a few more tricks.

It is possible to perform the FFT of two real functions at the same time by using the imaginary input values to the FFT for the second real function. There is then a post processing performed on the FFT results which separate the FFTs of the two functions. Using a similar procedure one can perform a transform on 2N real samples using an N complex sample transform.

The procedure involves alternating the real sample values between the real and imaginary inputs to the FFT. If, as in our example, the input to the FFT is a 2 by 32 array containing the complex values for 32 inputs, the 64 real samples would be loaded into it as follows:

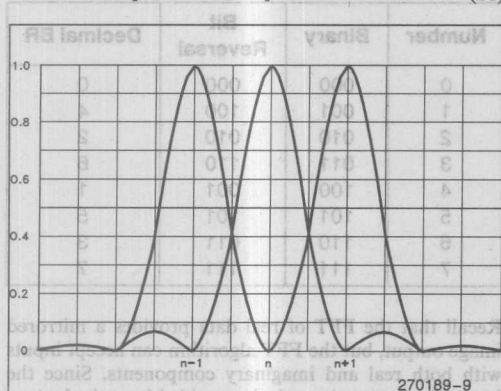
N	00 01 02 03 04 05 06 07 30 31
REAL	00 02 04 06 08 10 12 14 60 62
IMAGINARY	01 03 05 07 09 11 13 15 61 63

This procedure is referred to as a pre-weave. In order to derive the desired results, the FFT is run, and then a post-weave operation is performed. The formula for the post-weave is shown below:

$$X_r(n) = \left[\frac{R(n) + R(N-n)}{2} \right] + \cos \frac{\pi n}{N} \left[\frac{I(N) + I(N-n)}{2} \right]$$

$$X_i(n) = \left[\frac{I(n) - I(N-n)}{2} \right] + \sin \frac{\pi n}{N} \left[\frac{R(N) - R(N-n)}{2} \right]$$

$$n = 0, 1, \dots, N-1$$



(a.) Relative Power of Windows (Side Lobes of Side Bins Removed for Clarity).

Where $R(n)$ is the real FFT output value
 $I(n)$ is the imaginary FFT output value
 $X_r(n)$ is the real post-weave output
 $X_i(n)$ is the imaginary post-weave output

Note that the output is now one-sided instead of mirrored around the center frequency as it is in Figure 5. The magnitude of the signal at each frequency is calculated by taking the square root of the sum of the squares. The magnitude can now be plotted against frequency, where the frequency steps are defined as:

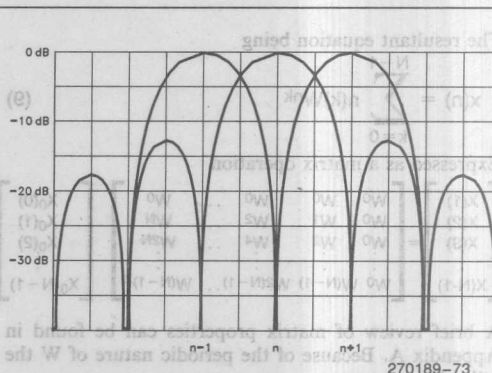
$$\frac{n}{NT} \quad n = 0, 1, 2, 3, \dots, N-1$$

Where N is the number of complex samples (ie. 32 in this case) T is the time between samples

A value of zero on the frequency scale corresponds to the DC component of the waveform. Most signal analysis is done using Decibels (dB), the conversion is $\text{dB} = 10 \text{ LOG (Magnitude squared)}$. Decibels are not used as an absolute measure, instead signals are compared by the difference in decibels. If the ratio between two signals is 1:2 then there will be a 3 dB difference in their power.

5.0 USING THE FFT

There are several things to be aware of when using FFTs, but with the proper cautions, the FFT output can be used just like that of a spectrum analyzer. The



(b.) 10 Log Relative Power of Windows (Side Lobes of Side Bins Removed for Clarity).

Figure 6. Bin Windows

first precaution is that the FFT is a discrete approximation to a continuous Fourier Transform, so the output will seldom fit the theoretical values exactly, but it will be very close.

Since the programs in this application note generate a one-sided transform with $N=32$, the frequency granularity is fairly coarse. Each of the frequency components output from the FFT is actually the sum of all energy within a narrow band centered on that frequency. This band of sensitivity is referred to as a "bin". The reported magnitude is the actual magnitude multiplied by the value of the bin window at the actual frequency. Figure 6 shows several bin windows. Note that these windows overlap, so that a frequency midway between the two center frequencies will be reported as energy split between both windows. Be careful not to

confuse the *sampling window* NT with *bin windows* or with the *windowing function*.

Another area of caution is the relationship of the sampling window to the frequency of the waveform. For the best accuracy, the window should cover an exact multiple of the period of the waveform being analyzed. If it covers less than one period, the results will be invalid. Other variations from ideal will not produce invalid results, just additional noise in the output.

If the sampling window does not cover an exact multiple of all of the frequency components of a waveform, the FFT results will be noisy. The reason for this is the sharp edge that the FFT sees when the edges of the window cut off the input waveform. Figure 7 shows a waveform that is an exact multiple of the window and

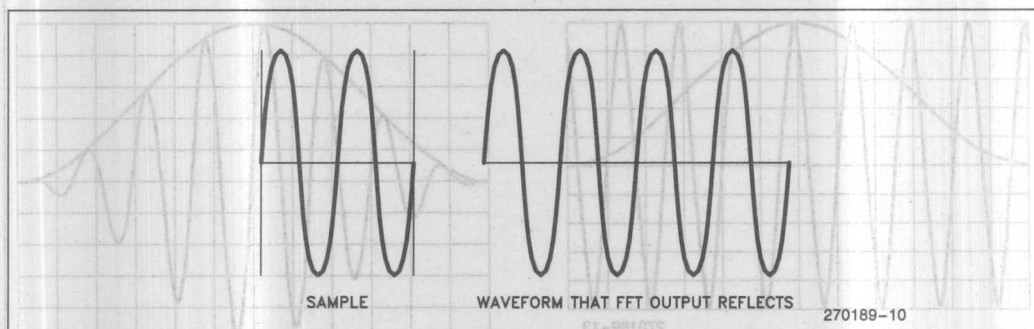


Figure 7. Waveform is a Multiple of the Window

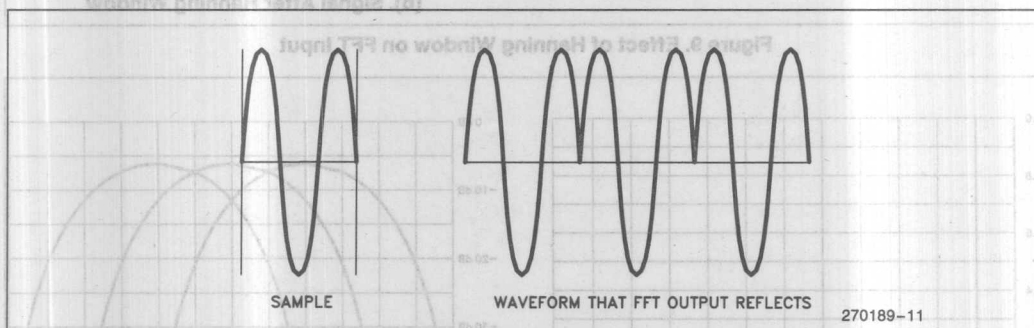


Figure 8. Waveform is Not a Multiple of the Window

the periodic waveform that the FFT output reflects. In Figure 8, the waveform is not a multiple of the window and the waveform that the FFT output reflects has discontinuities. These discontinuities contribute to the noise in an FFT output. This noise is called "spectral leakage", or simply "leakage", since it is leakage between one frequency spectrum and another which is caused by digitization of an analog process.

To reduce this leakage, a process called windowing is used. In this procedure the input data is multiplied by specific values before being used in the FFT. The term "windowing" is used because these values act as a window through which the input data passes. If the input window goes smoothly to zero at both endpoints of

the sampling window, there can be no discontinuities. Figure 9 shows a Hanning window and its effect on the input to an FFT. The Hanning window was named after its creator, Julius Von Hann, and is one of the most commonly used windows. More information on windowing and the types of windows can be found in the paper by Harris listed in the bibliography. As expected, the results of the FFT are changed because of the input windowing, but it is in a very predictable way.

Using the Hanning window results in bin windows which are wider and lower in magnitude than normal, as can be seen by comparing Figure 6 with Figure 10. For an input frequency which is equal to the center frequency of a bin window, the attenuation will be 6 dB on the center frequency. Since the bin windows are

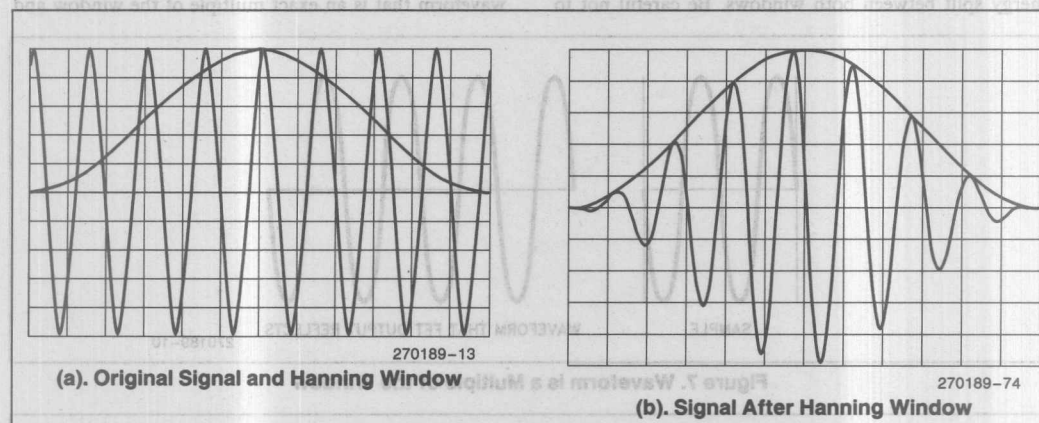


Figure 9. Effect of Hanning Window on FFT Input

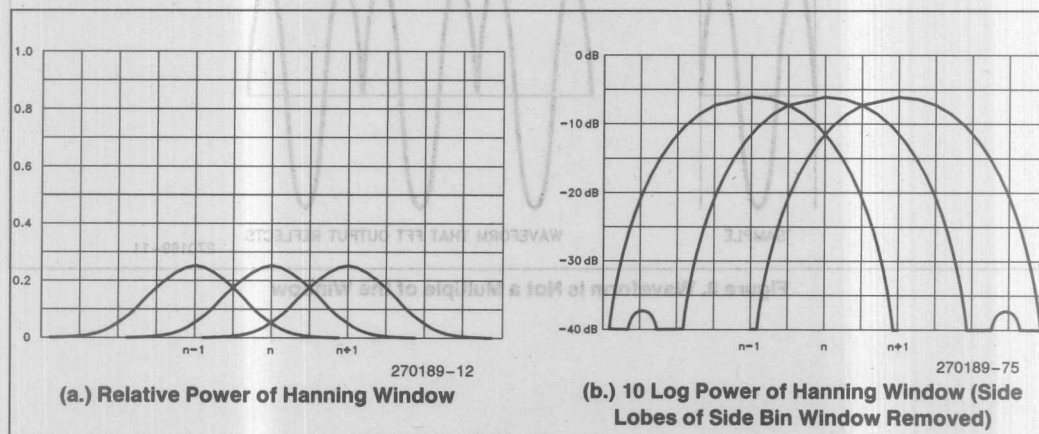


Figure 10. Bin Windows after Using Hanning Input Window

wider than normal, the input frequency will also have energy which falls into the bins on either side of center. These side bins will show a reading of 6 dB below the center window. The disadvantage of this spreading is far less than the advantage of removing leakage from the FFT output.

A set of FFT output plots are included in the Appendix. These plots show the effect of windowing on various signals. There are examples of all of the cases described above. A brief discussion of the plots is also presented.

Applications which can make use of this frequency magnitude information include a wide range of signal processing and detection tasks. Many of these tasks use digital filtering and signature analysis to match signals to a standard. This technique has been applied to anti-knock sensors for automobile engines, object identification for vision systems, cardiac arrhythmia detectors, noise separation and many other applications. The ability to do this on a single-chip computer opens a door to new products which would have not been possible or cost effective previously.

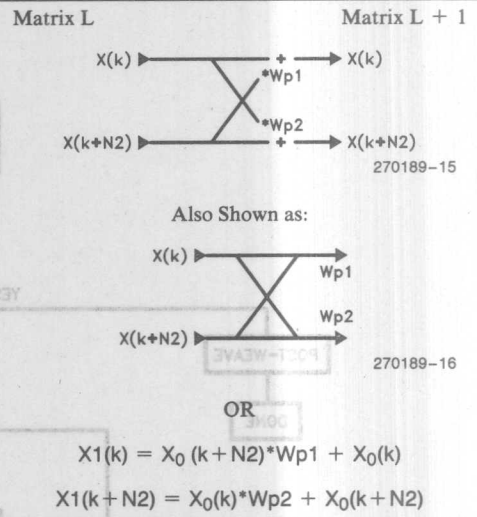
The next four sections of this application note cover the operation of the programs on a line by line basis. Section 6 shows an implementation of the FFT algorithm in BASIC. This code is used as a template to write the ASM96 code in Section 7. Sections 8, 9, and 10 cover the code sections which support the FFT module. After all of the code sections are discussed, an overview of how to use the program is presented in Section 11.

6.0 BASIC PROGRAM FOR FFTS

The algorithm for this FFT is shown in the flowchart in Figure 11 and the BASIC program in Listing 1. There are four sections to this program: initialization, pre-weaving, transform calculation, and post-weaving. The flowchart is generalized, however, the BASIC program has been optimized for assembly language conversion with 64 real samples.

On the flowchart, the initialization and pre-weaving sections are incorporated as "Read in Data". The data to be read includes the raw data as well as the size of the array and the scaling factor. The details for pre-weaving have been discussed earlier, and initialization varies from computer to computer. LOOP COUNT keeps track of which of the factored matrices are being multiplied. SHIFT is the shift count which is used to determine the power of W (as defined earlier) which will be used in the loop.

For each loop N calculations are performed in sets of two. Each calculation set is referred to as a butterfly and has the following form:



In general, the W factors are not the same. However, for the case of this FFT algorithm, Wp1 will always equal $(-Wp2)$. This is because of the way in which "p" is calculated, and the fact that W(x) is a sinusoidal function.

The inner loop in the flowchart is performed N2 times. For LOOP=1, N2=N/2 and if INCNT=N2 then $k=N2$ and $k+N2=N$, so the first loop is done and parameters LOOP, N2, and SHIFT are updated. For the first loop, all N/2 sets of calculations are performed contiguously. As LOOP increases, the number of contiguous calculations are cut in half, until LOOP=EXPONENT.

When LOOP=EXPONENT, N2=1, the butterfly is then performed on adjacent variables. Figure 12 shows the butterfly arrangement for a calculation where N=8, so that EXPONENT=3.

The BASIC program follows this flowchart, but operations have been grouped to make it easier to convert it to assembly language. Also not shown in the flowchart are several divide by 2 operations. There are five in the main section, one per loop. These provide the T/NT factor in equation 8 for N=32 ($2^5=32$). There is also an extra divide by two in the post-weave section. It is required to prevent overflows when performing the 16-bit signed arithmetic in the ASM96 program. As a result of these operations, the input scale factor is $\pm 1 = \pm 32767$ and the output scaling is $\pm 1 = \pm 16384$. Note, the maximum input values are ± 0.99997 .

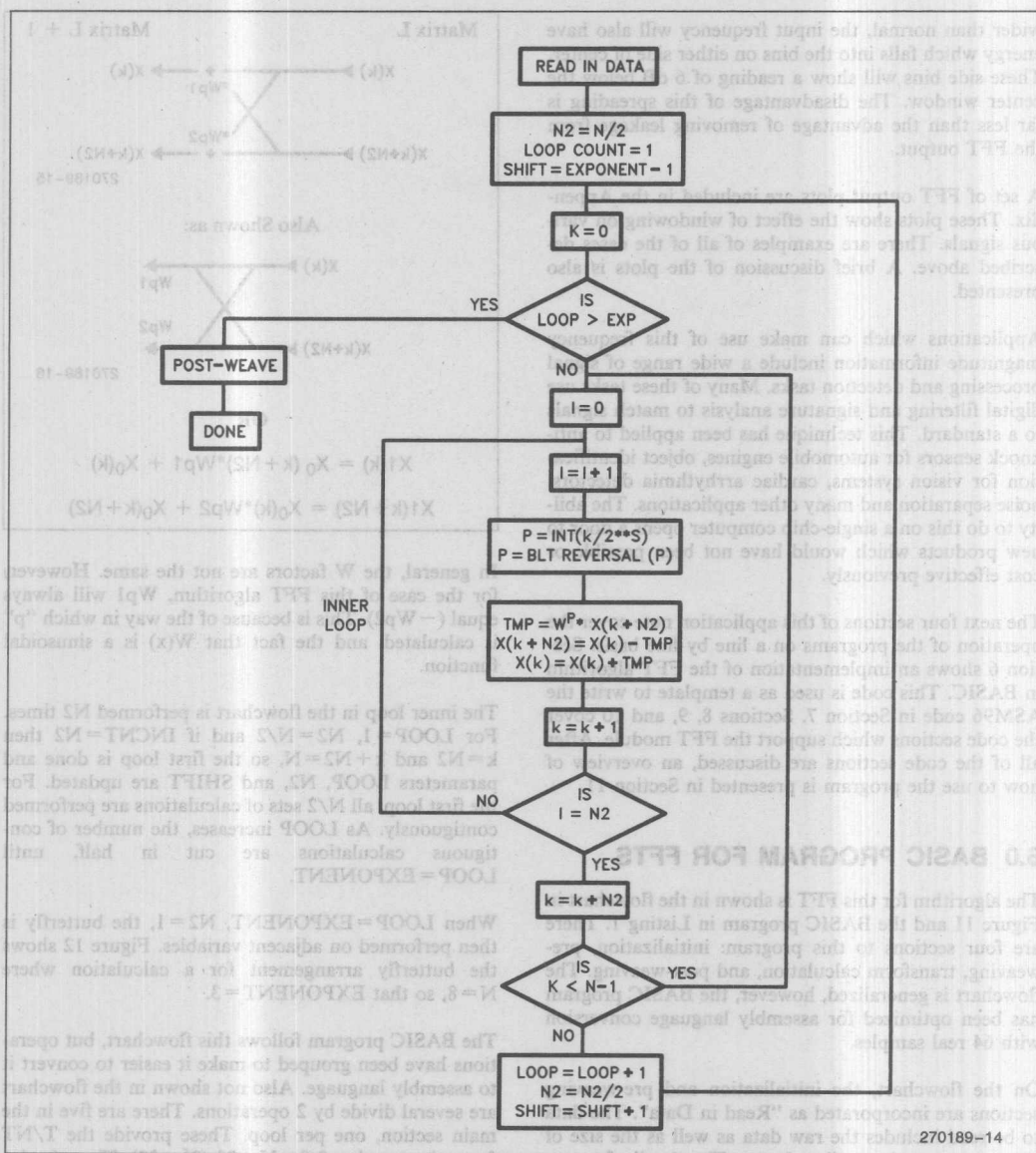


Figure 11. Flowchart of Basic Program

```

100 ' THIS IS FFT13, FEBRUARY 4, 1986
105 '
110 ' COPYRIGHT INTEL CORPORATION, 1985
115 ' BY IRA HORDEN, MCO APPLICATIONS
120 '
125 ' THIS PROGRAM PERFORMS A FAST FOURIER TRANSFORM ON 64 REAL DATA POINTS
130 ' USING A 2N-POINTS WITH AN N-POINT TRANSFORM ALGORITHM. THE FIRST
135 ' SECTION OF THE PROGRAM PERFORMS A STANDARD TRANSFORM ON DATA THAT HAS
140 ' BEEN INTERLEAVED BETWEEN THE REAL AND IMAGINARY INPUT VALUES. THE
145 ' RESULTS OF THAT TRANSFORM ARE THEN POST-PROCESSED IN THE SECOND SECTION
150 ' OF THE PROGRAM TO PROVIDE THE 32 OUTPUT BUCKETS. (THE OUTPUT VALUES ARE
155 ' MULTIPLIED BY "M" TO MAKE IT EASY TO COMPARE WITH THE ASM-96 PROGRAM
160 '
165 INPUT "NAME OF LIST FILE"; LST$
170 PRINT
175 OPEN LST$ FOR OUTPUT AS #1
180 '
190 ' SET UP VARIABLES FOR BASIC
200 '
210 DIM XR(32), XI(32), WR(32), WI(32), BR(32)
220 M=16383
230 N=32 : N1=31 : N2=N/2
240 LOOP=1 : K=0 : EXPONENT=5 : SHIFT=EXPONENT-1
250 PI=3.141592654# : TPN=2*PI/N : PIN=PI/N
260 '
270 ' READ IN CONSTANTS
280 FOR P=0 TO 31 : PN=P*TPN
290 WR(P)=COS(PN) : WI(P)=-SIN(PN) : READ BR(P)
300 NEXT P
310 '
320 FOR K=0 TO 31
330 READ XR(K) : READ XI(K)
340 NEXT K
350 '
360 '
370 ' INITIALIZATION OF LOOP
380 K=0
390 IF LOOP>EXPONENT THEN GOTO 700
400 INCONT=0
410 '
420 ' ACTUAL CALCULATIONS BEGIN HERE
430 INCONT=INCONT+1
440 P=BR(INT(K/(2^SHIFT)))
450 WRP=WR(P) : WIP=WI(P) : KN2=K+N2
460 TMPR= (WRP*XR(KN2) - WIP*XI(KN2))/2
470 TMPI= (WRP*XI(KN2) + WIP*XR(KN2))/2
480 TMPR1=XR(K)/2 : TMPI1=XI(K)/2
490 XR(K+N2) = TMPR1 - TMPR
500 XI(K+N2) = TMPI1 - TMPI
510 XR(K) = TMPR1 + TMPR
520 XI(K) = TMPI1 + TMPI
530 '
540 K=K+1
550 IF INCONT<N2 THEN GOTO 430
560 K=K+N2
570 IF K<N1 THEN GOTO 430
580 LOOP=LOOP+1 : N2=N2/2
590 SHIFT=SHIFT-1
600 GOTO 400
610 '
620 '
630 '
640 '
650 '
660 '
670 '
680 '
690 '
700 '

```

' SINCE THE ARRAY IS PROCESSED 2 POINTS AT A TIME,
 ' ONLY N/2 LOOPS NEED TO BE MADE. ON EACH PASS,
 ' THE VALUE OF N2 CHANGES AND SMALLER CONSECUTIVE
 ' SECTIONS ARE PROCESSED.

270189-17

Listing 1—BASIC FFT Program

```

694 '
695 '
696 '
697 '
700 ' POST-PROCESSING AND REORDERING BEGIN HERE
710 '
720 FORK = 0 TO 31
730 KPIN = K*PIN + 1
740 XBRK = XR(BR(K)) : XIBRK = XI(BR(K)) : CONDENSED FOR EASE OF ASM PROGRAMMING
750 XBRNK = XR(BR(N-K)) : XIBRNK = XI(BR(N-K))
760 TI = (XIBRK + XIBRNK) / 2
770 TR = (XBRBK - XBRBNK) / 2
780 XRT = (XBRBK + XBRBNK) / 4
790 XIT = (XIBRK - XIBRNK) / 4
800 OUTR = XRT + TI * COS(KPIN) / 2 - TR * SIN(KPIN) / 2
810 OUTI = XIT - TI * SIN(KPIN) / 2 - TR * COS(KPIN) / 2
820 '
830 MAGSQ = OUTR * OUTR + OUTI * OUTI ' THE ASM-96 PROGRAM USES A TABLE LOOK-UP
840 MAG = SQR(MAGSQ) ' ROUTINE TO CALCULATE SQUARE ROOTS
845 IF MAGSQ * M < .5 THEN DECIBEL = 0 : GOTO 900
847 DBFACT = M / 2 / 32767 * M * M * 2 / 64K
850 DECIBEL = 10 * LOG(MAGSQ * DBFACT)
860 DECIBEL = DECIBEL * .434294481
900 GOTO 930
910 PRINT #1, USING "##### "; K,
920 PRINT #1, USING "\ " ; HEX$(M*OUTR), HEX$(M*OUTI), HEX$(M*MAG);
930 ' GOTO 950
942 PRINT #1, USING "## "; K;
943 PRINT #1, USING "##### "; OUTR, OUTI, MAG;
945 PRINT #1, USING "####.### "; DECIBEL;
947 PRINT #1, USING "##### "; M*OUTR, M*OUTI, M*MAG
950 NEXT K
960 '
970 IF LST$(<)"SCRN:" THEN PRINT #1, CHR$(12)
999 END
1000 END
1010 ' DATA FOR BR(P) - BIT REVERSAL
1020 DATA 0,16,8,24,4,20,12,28,2,18,10,26,6,22,14,30
1030 DATA 1,17,9,25,5,21,13,29,3,19,11,27,7,23,15,31
1040 ' DATA FOR XR,XI
1050 DATA 2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2
1060 DATA 2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2
1070 DATA -2,-2,-2,-2,-2,-2,-2,-2,-2,-2,-2,-2,-2,-2,-2,-2
1080 DATA -2,-2,-2,-2,-2,-2,-2,-2,-2,-2,-2,-2,-2,-2,-2,-2

```

Listing 1—BASIC FFT Program (Continued)

Lines 165-175 set up the file for printing the data, this can be SCRIN:, LPT1:, or any other file.

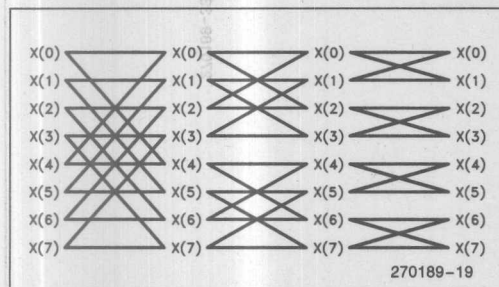


Figure 12. Butterflies with N=8

Lines 200-310 set up the constants and calculate the WP terms which are stored in the matrices WR(p) and WI(p), for the real and imaginary component respectively.

Lines 320-350 read in the data, alternately placing it into the real and imaginary arrays. The data is scaled by 2 to make the data table simpler.

Lines 410-430 initialize the loop and test for completion.

Lines 450-620 perform the FFT algorithm. Note that all calculations are complex, with the suffixes "R" and "I" indicating real and imaginary components respectively.

The variables on line 470, TMPR1 and TMP11 would normally not be used in a BASIC program as more than one operation can be performed on each line. However, indirect table lookups always use a separate line of assembly code, so separate lines have been used here.

Lines 700-810 perform the post-weave. This is not in the flowchart, but can be found in Equation 11. Once again, table look-ups are separated and additional variables are used for clarity. The variables BR(x) are the bit reversal values of x.

Line 830 calculates the magnitude of the harmonic components.

Lines 900-950 print the results of the calculations, with line 900 determining if the print-out should be in hex or decimal.

Lines 1000-1080 are the data for the bit reversal values and input datapoints. The input waveform is one cycle of a square-wave.

7.0 ASM96 PROGRAM FOR FFTS

The BASIC program just presented has been used as an outline for the ASM96 program shown in Listing 2. There are many advantages to using the BASIC program as a model, the main ones being debugging and testing. Since the BASIC program is so similar in program flow to the ASM96 program, it's possible to stop the ASM96 program at almost any point and verify that the results are correct.

MCS-96 MACRO ASSEMBLER FFT_RUN

02/18/86

PAGE 1

SERIES-III MCS-96 MACRO ASSEMBLER, V1.0

SOURCE FILE: :F2:FFTRUN.A96

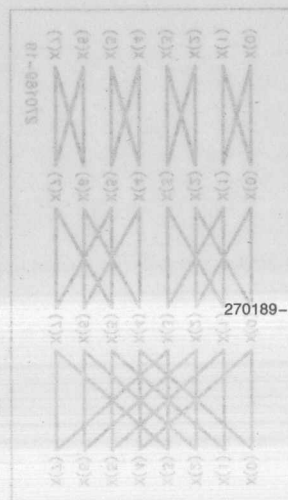
OBJECT FILE: :F2:FFTRUN.OBJ

CONTROLS SPECIFIED IN INVOCATION COMMAND: NOSB

```

ERR LOC OBJECT      LINE    SOURCE STATEMENT
1      $pagelength(50)
2
3      FFT_RUN MODULE STACKSIZE(6)
4
5      ; Intel Corporation, January 24, 1986
6      ; by Ira Horden, MCO Applications
7
8
9      ;
10     ; This module performs a fast fourier transform (FFT) on 64 real data
11     ; points using a 2N-point algorithm. The algorithm involves using a standard
12     ; FFT procedure for 32 real and 32 imaginary numbers. The real and imaginary
13     ; arrays are filled alternately with real data points, and the output of the
14     ; FFT is run through a post-processor. The result is a one-sided array with 32
15     ; output buckets. The post processing includes a table lookup algorithm for
16     ; taking the square root of an unsigned 32-bit number.
17
18     ; All of the calculations in the main FFT program are done using 16-bit
19     ; signed integers. The maximum value of any frequency component is therefore
20     ; +/- 32K. (Note that a square wave of +/-32K has a fundamental component
21     ; greater than +/- 40K). Wherever possible tables are used to increase the
22     ; speed of math operations. The complete transform, including obtaining the
23     ; absolute magnitude of each frequency component, executes in 12
24     ; milliseconds with internal variables, 14 ms with external.
25
26     ; The program requires two 32-word input arrays, with the sample values
27     ; alternated between the two. These start at KREAL and KIMAG. The resultant
28     ; magnitude will be placed in a 32-word array at FFT_OUT. These are all
29     ; externally defined variables. The external constant SCALE_FACTOR is used to
30     ; divide the output when averaging will be used. Since the program averages
31     ; its output, it is necessary to clear the array based at FFT_OUT before
32     ; calling FFT_CALC to start the program.
33
34     ; The program was originally written in BASIC for testing purposes. The
35     ; comments include these BASIC statements to make it easier to follow the
36     ; algorithm.
37     $EJECT

```



270189-33

```

MCS-96 MACRO ASSEMBLER    FFT_RUN                                02/18/86    PAGE    2

ERR LOC  OBJECT              LINE    SOURCE STATEMENT
0000                                           38
                                           39 RSEG
                                           40 EXTRN port1, zero, error
                                           41
0024                                           42 OSEG at 24H
0024                                           43 TMPR: dsl 1 ; Temporary register, Real
0028                                           44 TMPRI: dsl 1 ; Temporary register, Imaginary
002C                                           45 TMPRI: dsl 1 ; Temporary register, Real
0030                                           46 TMPRI: dsl 1 ; Temporary register, Imaginary
0034                                           47 XRTMP: dsl 1 ; Temporary data register, Real
0038                                           48 XRTMP: dsl 1 ; Temporary data register, Imaginary
003C                                           49 XRRK: dsl 1 ; Temporary register, Real
0040                                           50 XRRK: dsl 1 ; Temporary register, Imaginary
0044                                           51 XRRK: dsl 1 ; Temporary register, Real
0048                                           52 XRRK: dsl 1 ; Temporary register, Imaginary
003C                                           53 diff equ xrrk :long ; Table difference for square root
0040                                           54 sqrt equ xrrk :long ; Square root
0040                                           55 log equ xrrk :long ; 10 Log magnitude*2
0044                                           56 nxlloc equ xrrk :long ; Next location in table
003C                                           57 WRP equ xrrk :word ; Multiplication factor, Real
003E                                           58 WRP equ xrrk+2 :word ; Multiplication factor, Imaginary
0040                                           59 WRP equ xrrk :word ; Multiplication factor, Real
0042                                           60 WRP equ xrrk+2 :word ; Multiplication factor, Imaginary
0044                                           61 IN_CNT equ xrrk+2 :word ; n divided by 2 (0 < n < N) *2
0044                                           62 NDIV2 equ xrrk :word ; n divided by 2 (0 < n < N) *2
004C                                           63 KPTR: dsl 1 ; K for counter #2 to index words
004E                                           64 KN2: dsl 1 ; KPTR + NDIV2
0050                                           65 N_SUB_K: dsl 1 ; N-K #2 to index words
0052                                           66 RE: dsl 1 ; Bit reversed pointer of KPTR
0054                                           67 RNI: dsl 1 ; Bit reversed pointer of N_SUB_K
0056                                           68 SHFT_CNT: dsl 1 ; Shift counter
0058                                           69 LOOP_CNT: dsl 1 ; Loop counter
004E                                           70 ptr: equ kn2 :word ; Pointer for square root table
0000                                           71 DSEG
0000                                           72 EXTRN FFT_MODE ; FFT MODE: mode for FFT input and graphing
0000                                           73 EXTRN XREAL, XIMAG ; XREAL, XIMAG: Base addresses for 32 16-bit signed
0000                                           74 ; entries for real and imaginary numbers respectively.
0000                                           75 EXTRN FFT_OUT ; FFT_OUT: Starting address for 32 word array
0000                                           76 ; of magnitude information.
0000                                           77
0000                                           78
0000                                           79
0000                                           80 EXTRN OUTR: dsl 32 ; Real component of fft
0040                                           81 OUTI: dsl 32 ; Imaginary component of waveform
0000                                           82 PUBLIC OUTR,OUTI
0000                                           83
0000                                           84 $EJECT

```

270189-34

MCS-96 MACRO ASSEMBLER FFT_RUN 02/18/86 PAGE 3

```

ERR LOC OBJECT          LINE      SOURCE STATEMENT
      2280              85
      2280              86 CSEG at 2280H
      2280              87
      2280              88 PUBLIC fft_calc      ; Starting point for FFT algorithm
      2280              89
      2280              90 EXTRN scale_factor    ; Shift factor used to prevent overflow when averaging
      2280              91                      ; fft outputs
      2280              92
      2280              93
      2280              94
      2280              95 FFT_CALC: MOV     XREG     ; START FOURIER CALCULATIONS
      2280 1100          E 96      clrb     error      ; 400 ' INITIALIZATION OF LOOP
      2282 B10100        E 97      ldb     portl,#00000001b ;*** Indication Only
      2282              98
      2285 FC            99      clrvt     edn     ;
      2286 B10158        100     ldb     loop_cnt,#1
      2289 B10456        101     ldb     shift_cnt,#4
      229C A1200044      102     ld     ndiv2,#32
      2290              103
      2290              104 OUT_LOOP: MOV     XREG     ;
      2290 950400        E 105     xorb     portl,#00000100b ;*** Indication Only
      2293 014C          106     clr     kptr
      2293              107
      2295 990558        108     cmpb     loop_cnt,#5      ; 32=2^5
      2298 DA0220A3      109     bgt     UNWEAVE
      2298              110
      229C              111 MID_LOOP: MOV     XREG     ;
      229C 0142          112     clr     in_cnt      ; 430 INCNT=0
      229C              113
      229C              114
      229C              115
      229E              116 IN_LOOP: MOV     XREG     ;
      229E 65020042      117     add     in_cnt,#2      ; 450 INCNT=INCNT+1
      229E              118
      22A2 A04C40        119     ld     pwr,kptr      ; 460 P=BR(INT(K/(2*SHIFT)))
      22A5 085640        120     shr     pwr,shift_cnt
      22A8 71FE40        121     andb     pwr,#11111110B
      22AB A341003840     122     ld     pwr,brev[pwr]
      22AB              123
      22B0 A34144393C     124     gw: ld     wip,wr[pwr]
      22B5 A34186393E     125     ld     wip,wi[pwr]
      22BA 44444C4E      126     add     kn2,kptr,ndiv2
      22BA              127     $eject

```

270189-35

270189-36

ERR	LOC	OBJECT	LINE	SOURCE STATEMENT
			165 ;	560 K=K+1
	2318	6502004C	166 ik: add	kptr,#2
			167	
			168 ;	570 IF INCNT<N2 THEN GOTO 450
	231C	884442	169	cmp in_cnt,ndiv2
	231F	D602277B	170 ! blt	IN_LOOP
			171	
			172 ;	580 K=K+N2
	2323	64444C	173	add kptr,ndiv2
			174 ;	590 IF K<N1 THEN GOTO 430
	2326	893E004C	175	cmp kptr,#62
	232A	D602276E	176 ! blt	MID_LOOP
			177	
			178	
	232E	1758	179	incb loop_cnt
	2330	0A0144	180	shra ndiv2,#1
	2333	1556	181	decb shft_cnt
			182 ;	610 GOTO 400
	2335	2759	183	br OUT_LOOP
			184	
			185	
	2337	B10100	186	ERR1: ldb error,#01
	233A	F0	187	ret
	233B	B10200	188	ERR2: ldb error,#02
	233E	F0	189	ret
			190	
			191	\$EJECT
			192	
			193	
			194	
			195	
			196	
			197	
			198	
			199	
			200	
			201	
			202	
			203	
			204	
			205	
			206	
			207	
			208	
			209	
			210	
			211	
			212	
			213	
			214	
			215	
			216	
			217	
			218	
			219	
			220	
			221	
			222	
			223	
			224	
			225	
			226	
			227	
			228	
			229	
			230	
			231	
			232	
			233	
			234	
			235	
			236	
			237	
			238	
			239	
			240	
			241	
			242	
			243	
			244	
			245	
			246	
			247	
			248	
			249	
			250	
			251	
			252	
			253	
			254	
			255	
			256	
			257	
			258	
			259	
			260	
			261	
			262	
			263	
			264	
			265	
			266	
			267	
			268	
			269	
			270	
			271	
			272	
			273	
			274	
			275	
			276	
			277	
			278	
			279	
			280	
			281	
			282	
			283	
			284	
			285	
			286	
			287	
			288	
			289	
			290	
			291	
			292	
			293	
			294	
			295	
			296	
			297	
			298	
			299	
			300	
			301	
			302	
			303	
			304	
			305	
			306	
			307	
			308	
			309	
			310	
			311	
			312	
			313	
			314	
			315	
			316	
			317	
			318	
			319	
			320	
			321	
			322	
			323	
			324	
			325	
			326	
			327	
			328	
			329	
			330	
			331	
			332	
			333	
			334	
			335	
			336	
			337	
			338	
			339	
			340	
			341	
			342	
			343	
			344	
			345	
			346	
			347	
			348	
			349	
			350	
			351	
			352	
			353	
			354	
			355	
			356	
			357	
			358	
			359	
			360	
			361	
			362	
			363	
			364	
			365	
			366	
			367	
			368	
			369	
			370	
			371	
			372	
			373	
			374	
			375	
			376	
			377	
			378	
			379	
			380	
			381	
			382	
			383	
			384	
			385	
			386	
			387	
			388	
			389	
			390	
			391	
			392	
			393	
			394	
			395	
			396	
			397	
			398	
			399	
			400	
			401	
			402	
			403	
			404	
			405	
			406	
			407	
			408	
			409	
			410	
			411	
			412	
			413	
			414	
			415	
			416	
			417	
			418	
			419	
			420	
			421	
			422	
			423	
			424	
			425	
			426	
			427	
			428	
			429	
			430	
			431	
			432	
			433	
			434	
			435	
			436	
			437	
			438	
			439	
			440	
			441	
			442	
			443	
			444	
			445	
			446	
			447	
			448	
			449	
			450	
			451	
			452	
			453	
			454	
			455	
			456	
			457	
			458	
			459	
			460	
			461	
			462	
			463	
			464	
			465	
			466	
			467	
			468	
			469	
			470	
			471	
			472	
			473	
			474	
			475	
			476	
			477	
			478	
			479	
			480	
			481	
			482	
			483	
			484	
			485	
			486	
			487	
			488	
			489	
			490	
			491	
			492	
			493	

270189-38

270189-39

Listing 2—ASM96 FFT Program (Continued)

6-124


```

MCS-96 MACRO ASSEMBLER    FFT_RUN                                02/18/86    PAGE    8

ERR LOC OBJECT            LINE    SOURCE STATEMENT

276
277                ;;;; *** CALCULATE 10 log magnitude^2 ***
278    ; Output = 512*10*LOG(x)  x=1,2,3 ... 64K
279
23F7 0156            280    CALC_LOG:
23F9 0F5624          281        clr     shift_cnt
23FC 990F56          282        normal tmp,shift_cnt ; Normalize and get normalization factor
23FF DA04            283        cmpb    shift_cnt,$15
284        jle     LOG_IN_RANGE ; Jump if SHIFT_CNT <= 15
285
2401 0140            286        clr     log
2403 202C            287        br      LOG_STORE
288
2405 04565656        289    LOG_IN_RANGE:
2405 44565656        290        add     shift_cnt,shift_cnt,shift_cnt ; Make shift_cnt a pointer
291
2409 AC274E          292        ldbze   ptr,tmp+3 ; Most significant byte is table pointer
240C 444E4E4E        293        ptr,ptr,ptr
2410 65083A4E        294        add     ptr,$ LOG_TABLE-256 ; ptr= Table + offset (offset=tmp+3)
295        ; Use -256 since tmp+3 is always >= 128
2414 A24F40          296        ld      log,[ptr]+
2417 A24E44          297        ld      nxtloc,[ptr] ; Linear Interpolation
298
241A 684044          299        sub     nxtloc,log ; nxtloc = next log - log
300
241D AC263C          301        ldbze   diff,tmp+2 ; diff+1 = nxtloc * tmp+2 / 256
2420 6C443C          302        mulu    diff,nxtloc
303
2423 0C083C          304        shr     diff,$8 ; log = log + diff/256
2426 643C40          305        add     log,diff
2429 080540          306        shr     log,$5 ; 8192/32 * 20LOG(x) = 256 * 20LOG(x)
307
242C A7570A3C40      308        addc    log,log_offset[shift_cnt] ; add log of normalization factor
309
310        ; Log (M*N) = Log M + Log N
311
2431 0800040         E 312    LOG_STORE:
2431 0800040         E 313        shr     log,$SCALE_FACTOR
2434 A40040         E 314        addc    log,zero ; Divide to prevent overflow during
2437 674D0000040    E 315        add     log,FFT_OUT[kptr] ; averaging of outputs
243C C34D0000040    E 316        st      log,FFT_OUT[kptr]
317
2441 204500         E 318        BR      ENDL
319    $eject
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000

```

270189-40

intel

AP-275

1611

```

MCS-96 MACRO ASSEMBLER    FFT_RUN                                02/18/86    PAGE    9

ERR LOC  OBJECT                LINE    SOURCE STATEMENT
2443                                320
2443                                321      CALC_SQRT:      ;;;; *** CALCULATE SQUARE ROOT ***
2443                                322
2443 0156                        323      clr      shift_cnt
2445 0F5624                      324      norml   tmpr,shift_cnt ; Normalize and get normalization factor
2448 D705                        325
244A C04200                      326      select  jne      SQRT_IN_RANGE ; Jump if tmpr > 0
244D 2029                        327      st      zero,sqrt+2
244D 2029                        328      br      SQRT_STORE
244F 61000000                     329
244F 61000000                     330      SQRT_IN_RANGE:
244F 61000000                     331      ldbze   ptr,tmpr+3 ; Most significant byte is table pointer
2452 444E4E4E                     332      add     ptr,ptr,ptr
2456 6508394E                     333      add     ptr,#SQ_TABLE-256 ; ptr= Table + offset (offset=tmpr+3)
2456 6508394E                     334      ; Use -256 since tmpr+3 is always >= 128
245A A24F40                       335      ld      sqrt,[ptr]+
245D A24E44                       336      ld      nxtloc,[ptr] ; Linear Interpolation
2460 684044                       337
2460 684044                       338      sub     nxtloc,sqrt ; nxtloc = sqrt - next sqrt
2463 AC263C                       339      sgt     diff,tmpr+2 ; diff+1 = nxtloc * tmpr+2 / 256
2466 6C443C                       340      mulu    diff,nxtloc
2466 6C443C                       341
2469 AC3D3C                       342      ldbze   diff,diff+1 ; sqrt = sqrt + delta (diff < 0FFFH)
246C 643C40                       343      add     sqrt,diff
246F 44565656                     344
246F 44565656                     345      add     shift_cnt,shift_cnt,shift_cnt
2473 6F57C83940                   346
2473 6F57C83940                   347      mulu    sqrt,tab_sqrt[shift_cnt] ; divide by normalization factor
2473 6F57C83940                   348
2473 6F57C83940                   349      ;
2473 6F57C83940                   350      ; mulu acts as divide since if tab2=0FFFFH
2473 6F57C83940                   351      ; mulu acts as divide since if tab2=0FFFFH
2478 080042                       352      SQRT_STORE:
2478 080042                       353      shr     sqrt+2,#SCALE_FACTOR
247B A40042                       354      addc    sqrt+2,zero ; Divide to prevent overflow during
247E 674D000042                   355      add     sqrt+2,FFT_OUT[kptr] ; averaging of outputs
2483 C34D000042                   356      add     sqrt+2,FFT_OUT[kptr]
2483 C34D000042                   357
2483 C34D000042                   358      ;
2483 C34D000042                   359      ;
2483 C34D000042                   360      ;
2488 6502004C                     361      ENDL:      add     kptr,#2
248C 69020050                     362      sub     n_sub_k,#2
2490 D70226B4                     363      bne     UN_LOOP
2494 F0                           364      RET
2494 F0                           365
2494 F0                           366      $eject

```

270189-41

MCS-96 MACRO ASSEMBLER FFT_RUN 02/18/86 PAGE 10

```

ERR LOC OBJECT      LINE      SOURCE STATEMENT
3800      367      ;$nolist
3800      368      CSEG AT 3800H      ;;;      Use 2k for tables
3800      369
3800      370      BREV:      ; 2*bit reversal value
3800      371
3800      372      DCW      2*0, 2*16, 2*8,      2*24, 2*4, 2*20, 2*12, 2*28
3810      373      DCW      2*2, 2*18, 2*10, 2*26, 2*6, 2*22, 2*14, 2*30
3820      374      DCW      2*1, 2*17, 2*9,      2*25, 2*5, 2*21, 2*13, 2*29
3830      375      DCW      2*3, 2*19, 2*11, 2*27, 2*7, 2*23, 2*15, 2*31
3800      376
3840      377      SINFN:
3840      378      DCW      0,      3212,      6393,      9512,      12539,      15446,      18204,      20787
3850      379      DCW      23170,      25329,      27245,      28898,      30273,      31356,      32137,      32609
3860      380      DCW      32767,      32609,      32137,      31356,      30273,      28898,      27245,      25329
3870      381      DCW      23170,      20787,      18204,      15446,      12539,      9512,      6393,      3212
3880      382      DCW      0,      -3212,      -6393,      -9512,      -12539,      -15446,      -18204,      -20787
3890      383      DCW      -23170,      -25329,      -27245,      -28898,      -30273,      -31356,      -32137,      -32609
3900      384      DCW      -32767,      -32609,      -32137,      -31356,      -30273,      -28898,      -27245,      -25329
3910      385      DCW      -23170,      -20787,      -18204,      -15446,      -12539,      -9512,      -6393,      -3212
3920      386      DCW      0
3930      387
3940      388      COSFN:
3940      389      DCW      32767,      32609,      32137,      31356,      30273,      28898,      27245,      25329
3950      390      DCW      23170,      20787,      18204,      15446,      12539,      9512,      6393,      3212
3960      391      DCW      0,      -3212,      -6393,      -9512,      -12539,      -15446,      -18204,      -20787
3970      392      DCW      -23170,      -25329,      -27245,      -28898,      -30273,      -31356,      -32137,      -32609
3980      393      DCW      -32767,      -32609,      -32137,      -31356,      -30273,      -28898,      -27245,      -25329
3990      394      DCW      -23170,      -20787,      -18204,      -15446,      -12539,      -9512,      -6393,      -3212
4000      395      DCW      0,      3212,      6393,      9512,      12539,      15446,      18204,      20787
4010      396      DCW      23170,      25329,      27245,      28898,      30273,      31356,      32137,      32609
4020      397      DCW      32767
4030      398
4040      399      WR:      ;;;      WR = COS(K*2PI/N)
4050      400      DCW      32767,      32137,      30273,      27245,      23170,      18204,      12539,      6393
4060      401      DCW      0,      -6393,      -12539,      -18204,      -23170,      -27245,      -30273,      -32137
4070      402      DCW      -32767,      -32137,      -30273,      -27245,      -23170,      -18204,      -12539,      -6393
4080      403      DCW      0,      6393,      12539,      18204,      23170,      27245,      30273,      32137
4090      404      DCW      32767
4100      405
4110      406      WI:      ;;;      WI = -SIN(K*2PI/N)
4120      407      DCW      0,      -6393,      -12539,      -18204,      -23170,      -27245,      -30273,      -32137
4130      408      DCW      -32767,      -32137,      -30273,      -27245,      -23170,      -18204,      -12539,      -6393
4140      409      DCW      0,      6393,      12539,      18204,      23170,      27245,      30273,      32137
4150      410      DCW      32767,      32137,      30273,      27245,      23170,      18204,      12539,      6393
4160      411      DCW      0
4170      412      $ject:

```

270189-42

```

ERR LOC OBJECT LINE SOURCE STATEMENT
413
414
39C8 415 TAB_SQR: ; 65535/(square root of 2**SHFT_CNT) ; 0<=SHFT_CNT<32
39C8 416
39C8 417 ;; 1 2 4 8 16 32 64 128
39C8 418 DCW 65535, 46340, 32768, 23170, 16384, 11585, 8192, 5793
39C8 419
39D8 420 ;; 256 512 1024 2048 4096 8192 16384 32768
39D8 421 DCW 4096, 2896, 2048, 1448, 1024, 724, 512, 362
39D8 422
39E8 423 ;; 65536, 131072, 262144, 524288, ...
39F8 424 DCW 256, 181, 128, 91, 64, 45, 32, 23
39F8 425 DCW 16, 11, 8, 6, 4, 3, 2, 1
39F8 426
3A08 427
3A08 428 SQ_TABLE: ; square root of n * 2**24 N=128, 129, 130 ... 255
3A08 429
3A08 430 DCW 46341, 46522, 46702, 46881, 47059, 47237, 47415, 47591
3A18 431 DCW 47767, 47942, 48117, 48291, 48465, 48637, 48809, 48981
3A28 432 DCW 49152, 49322, 49492, 49661, 49830, 49998, 50166, 50332
3A38 433 DCW 50499, 50665, 50830, 50995, 51159, 51323, 51486, 51649
3A48 434 DCW 51811, 51972, 52134, 52294, 52454, 52614, 52773, 52932
3A58 435 DCW 53090, 53248, 53405, 53562, 53719, 53874, 54030, 54185
3A68 436 DCW 54340, 54494, 54647, 54801, 54954, 55106, 55258, 55410
3A78 437 DCW 55561, 55712, 55862, 56012, 56162, 56311, 56459, 56608
3A88 438 DCW 56756, 56903, 57051, 57198, 57344, 57490, 57636, 57781
3A98 439 DCW 57926, 58071, 58215, 58359, 58503, 58646, 58789, 58931
3AAB 440 DCW 59073, 59215, 59357, 59498, 59639, 59779, 59919, 60059
3ABB 441 DCW 60199, 60338, 60477, 60615, 60754, 60891, 61029, 61166
3AC8 442 DCW 61303, 61440, 61576, 61712, 61848, 61984, 62119, 62254
3AD8 443 DCW 62388, 62523, 62657, 62790, 62924, 63057, 63190, 63323
3AE8 444 DCW 63455, 63587, 63719, 63850, 63982, 64113, 64243, 64374
3AF8 445 DCW 64504, 64634, 64763, 64893, 65022, 65151, 65280, 65408
3AF8 446
3AF8 447 $eject
3B08 448
3B08 449
3B08 450
3B08 451
3B08 452
3B08 453
3B08 454
3B08 455
3B08 456
3B08 457
3B08 458
3B08 459
3B08 460
3B08 461
3B08 462
3B08 463
3B08 464
3B08 465
3B08 466
3B08 467
3B08 468
3B08 469
3B08 470
3B08 471
3B08 472
3B08 473
3B08 474
3B08 475
3B08 476
3B08 477
3B08 478
3B08 479
3B08 480
3B08 481
3B08 482
3B08 483
3B08 484
3B08 485
3B08 486
3B08 487
3B08 488
3B08 489
3B08 490
3B08 491
3B08 492
3B08 493
3B08 494
3B08 495
3B08 496
3B08 497
3B08 498
3B08 499
3B08 500
3B08 501
3B08 502
3B08 503
3B08 504
3B08 505
3B08 506
3B08 507
3B08 508
3B08 509
3B08 510
3B08 511
3B08 512
3B08 513
3B08 514
3B08 515
3B08 516
3B08 517
3B08 518
3B08 519
3B08 520
3B08 521
3B08 522
3B08 523
3B08 524
3B08 525
3B08 526
3B08 527
3B08 528
3B08 529
3B08 530
3B08 531
3B08 532
3B08 533
3B08 534
3B08 535
3B08 536
3B08 537
3B08 538
3B08 539
3B08 540
3B08 541
3B08 542
3B08 543
3B08 544
3B08 545
3B08 546
3B08 547
3B08 548
3B08 549
3B08 550
3B08 551
3B08 552
3B08 553
3B08 554
3B08 555
3B08 556
3B08 557
3B08 558
3B08 559
3B08 560
3B08 561
3B08 562
3B08 563
3B08 564
3B08 565
3B08 566
3B08 567
3B08 568
3B08 569
3B08 570
3B08 571
3B08 572
3B08 573
3B08 574
3B08 575
3B08 576
3B08 577
3B08 578
3B08 579
3B08 580
3B08 581
3B08 582
3B08 583
3B08 584
3B08 585
3B08 586
3B08 587
3B08 588
3B08 589
3B08 590
3B08 591
3B08 592
3B08 593
3B08 594
3B08 595
3B08 596
3B08 597
3B08 598
3B08 599
3B08 600
3B08 601
3B08 602
3B08 603
3B08 604
3B08 605
3B08 606
3B08 607
3B08 608
3B08 609
3B08 610
3B08 611
3B08 612
3B08 613
3B08 614
3B08 615
3B08 616
3B08 617
3B08 618
3B08 619
3B08 620
3B08 621
3B08 622
3B08 623
3B08 624
3B08 625
3B08 626
3B08 627
3B08 628
3B08 629
3B08 630
3B08 631
3B08 632
3B08 633
3B08 634
3B08 635
3B08 636
3B08 637
3B08 638
3B08 639
3B08 640
3B08 641
3B08 642
3B08 643
3B08 644
3B08 645
3B08 646
3B08 647
3B08 648
3B08 649
3B08 650
3B08 651
3B08 652
3B08 653
3B08 654
3B08 655
3B08 656
3B08 657
3B08 658
3B08 659
3B08 660
3B08 661
3B08 662
3B08 663
3B08 664
3B08 665
3B08 666
3B08 667
3B08 668
3B08 669
3B08 670
3B08 671
3B08 672
3B08 673
3B08 674
3B08 675
3B08 676
3B08 677
3B08 678
3B08 679
3B08 680
3B08 681
3B08 682
3B08 683
3B08 684
3B08 685
3B08 686
3B08 687
3B08 688
3B08 689
3B08 690
3B08 691
3B08 692
3B08 693
3B08 694
3B08 695
3B08 696
3B08 697
3B08 698
3B08 699
3B08 700
3B08 701
3B08 702
3B08 703
3B08 704
3B08 705
3B08 706
3B08 707
3B08 708
3B08 709
3B08 710
3B08 711
3B08 712
3B08 713
3B08 714
3B08 715
3B08 716
3B08 717
3B08 718
3B08 719
3B08 720
3B08 721
3B08 722
3B08 723
3B08 724
3B08 725
3B08 726
3B08 727
3B08 728
3B08 729
3B08 730
3B08 731
3B08 732
3B08 733
3B08 734
3B08 735
3B08 736
3B08 737
3B08 738
3B08 739
3B08 740
3B08 741
3B08 742
3B08 743
3B08 744
3B08 745
3B08 746
3B08 747
3B08 748
3B08 749
3B08 750
3B08 751
3B08 752
3B08 753
3B08 754
3B08 755
3B08 756
3B08 757
3B08 758
3B08 759
3B08 760
3B08 761
3B08 762
3B08 763
3B08 764
3B08 765
3B08 766
3B08 767
3B08 768
3B08 769
3B08 770
3B08 771
3B08 772
3B08 773
3B08 774
3B08 775
3B08 776
3B08 777
3B08 778
3B08 779
3B08 780
3B08 781
3B08 782
3B08 783
3B08 784
3B08 785
3B08 786
3B08 787
3B08 788
3B08 789
3B08 790
3B08 791
3B08 792
3B08 793
3B08 794
3B08 795
3B08 796
3B08 797
3B08 798
3B08 799
3B08 800
3B08 801
3B08 802
3B08 803
3B08 804
3B08 805
3B08 806
3B08 807
3B08 808
3B08 809
3B08 810
3B08 811
3B08 812
3B08 813
3B08 814
3B08 815
3B08 816
3B08 817
3B08 818
3B08 819
3B08 820
3B08 821
3B08 822
3B08 823
3B08 824
3B08 825
3B08 826
3B08 827
3B08 828
3B08 829
3B08 830
3B08 831
3B08 832
3B08 833
3B08 834
3B08 835
3B08 836
3B08 837
3B08 838
3B08 839
3B08 840
3B08 841
3B08 842
3B08 843
3B08 844
3B08 845
3B08 846
3B08 847
3B08 848
3B08 849
3B08 850
3B08 851
3B08 852
3B08 853
3B08 854
3B08 855
3B08 856
3B08 857
3B08 858
3B08 859
3B08 860
3B08 861
3B08 862
3B08 863
3B08 864
3B08 865
3B08 866
3B08 867
3B08 868
3B08 869
3B08 870
3B08 871
3B08 872
3B08 873
3B08 874
3B08 875
3B08 876
3B08 877
3B08 878
3B08 879
3B08 880
3B08 881
3B08 882
3B08 883
3B08 884
3B08 885
3B08 886
3B08 887
3B08 888
3B08 889
3B08 890
3B08 891
3B08 892
3B08 893
3B08 894
3B08 895
3B08 896
3B08 897
3B08 898
3B08 899
3B08 900
3B08 901
3B08 902
3B08 903
3B08 904
3B08 905
3B08 906
3B08 907
3B08 908
3B08 909
3B08 910
3B08 911
3B08 912
3B08 913
3B08 914
3B08 915
3B08 916
3B08 917
3B08 918
3B08 919
3B08 920
3B08 921
3B08 922
3B08 923
3B08 924
3B08 925
3B08 926
3B08 927
3B08 928
3B08 929
3B08 930
3B08 931
3B08 932
3B08 933
3B08 934
3B08 935
3B08 936
3B08 937
3B08 938
3B08 939
3B08 940
3B08 941
3B08 942
3B08 943
3B08 944
3B08 945
3B08 946
3B08 947
3B08 948
3B08 949
3B08 950
3B08 951
3B08 952
3B08 953
3B08 954
3B08 955
3B08 956
3B08 957
3B08 958
3B08 959
3B08 960
3B08 961
3B08 962
3B08 963
3B08 964
3B08 965
3B08 966
3B08 967
3B08 968
3B08 969
3B08 970
3B08 971
3B08 972
3B08 973
3B08 974
3B08 975
3B08 976
3B08 977
3B08 978
3B08 979
3B08 980
3B08 981
3B08 982
3B08 983
3B08 984
3B08 985
3B08 986
3B08 987
3B08 988
3B08 989
3B08 990
3B08 991
3B08 992
3B08 993
3B08 994
3B08 995
3B08 996
3B08 997
3B08 998
3B08 999
3B08 1000
3B08 1001
3B08 1002
3B08 1003
3B08 1004
3B08 1005
3B08 1006
3B08 1007
3B08 1008
3B08 1009
3B08 1010
3B08 1011
3B08 1012
3B08 1013
3B08 1014
3B08 1015
3B08 1016
3B08 1017
3B08 1018
3B08 1019
3B08 1020
3B08 1021
3B08 1022
3B08 1023
3B08 1024
3B08 1025
3B08 1026
3B08 1027
3B08 1028
3B08 1029
3B08 1030
3B08 1031
3B08 1032
3B08 1033
3B08 1034
3B08 1035
3B08 1036
3B08 1037
3B08 1038
3B08 1039
3B08 1040
3B08 1041
3B08 1042
3B08 1043
3B08 1044
3B08 1045
3B08 1046
3B08 1047
3B08 1048
3B08 1049
3B08 1050
3B08 1051
3B08 1052
3B08 1053
3B08 1054
3B08 1055
3B08 1056
3B08 1057
3B08 1058
3B08 1059
3B08 1060
3B08 1061
3B08 1062
3B08 1063
3B08 1064
3B08 1065
3B08 1066
3B08 1067
3B08 1068
3B08 1069
3B08 1070
3B08 1071
3B08 1072
3B08 1073
3B08 1074
3B08 1075
3B08 1076
3B08 1077
3B08 1078
3B08 1079
3B08 1080
3B08 1081
3B08 1082
3B08 1083
3B08 1084
3B08 1085
3B08 1086
3B08 1087
3B08 1088
3B08 1089
3B08 1090
3B08 1091
3B08 1092
3B08 1093
3B08 1094
3B08 1095
3B08 1096
3B08 1097
3B08 1098
3B08 1099
3B08 1100
3B08 1101
3B08 1102
3B08 1103
3B08 1104
3B08 1105
3B08 1106
3B08 1107
3B08 1108
3B08 1109
3B08 1110
3B08 1111
3B08 1112
3B08 1113
3B08 1114
3B08 1115
3B08 1116
3B08 1117
3B08 1118
3B08 1119
3B08 1120
3B08 1121
3B08 1122
3B08 1123
3B08 1124
3B08 1125
3B08 1126
3B08 1127
3B08 1128
3B08 1129
3B08 1130
3B08 1131
3B08 1132
3B08 1133
3B08 1134
3B08 1135
3B08 1136
3B08 1137
3B08 1138
3B08 1139
3B08 1140
3B08 1141
3B08 1142
3B08 1143
3B08 1144
3B08 1145
3B08 1146
3B08 1147
3B08 1148
3B08 1149
3B08 1150
3B08 1151
3B08 1152
3B08 1153
3B08 1154
3B08 1155
3B08 1156
3B08 1157
3B08 1158
3B08 1159
3B08 1160
3B08 1161
3B08 1162
3B08 1163
3B08 1164
3B08 1165
3B08 1166
3B08 1167
3B08 1168
3B08 1169
3B08 1170
3B08 1171
3B08 1172
3B08 1173
3B08 1174
3B08 1175
3B08 1176
3B08 1177
3B08 1178
3B08 1179
3B08 1180
3B08 1181
3B08 1182
3B08 1183
3B08 1184
3B08 1185
3B08 1186
3B08 1187
3B08 1188
3B08 1189
3B08 1190
3B08 1191
3B08 1192
3B08 1193
3B08 1194
3B08 1195
3B08 1196
3B08 1197
3B08 1198
3B08 1199
3B08 1200
3B08 1201
3B08 1202
3B08 1203
3B08 1204
3B08 1205
3B08 1206
3B08 1207
3B08 1208
3B08 1209
3B08 1210
3B08 1211
3B08 1212
3B08 1213
3B08 1214
3B08 1215
3B08 1216
3B08 1217
3B08 1218
3B08 1219
3B08 1220
3B08 1221
3B08 1222
3B08 1223
3B08 1224
3B08 1225
3B08 1226
3B08 1227
3B08 1228
3B08 1229
3B08 1230
3B08 1231
3B08 1232
3B08 1233
3B08 1234
3B08 1235
3B08 1236
3B08 1237
3B08 1238
3B08 1239
3B08 1240
3B08 1241
3B08 1242
3B08 1243
3B08 1244
3B08 1245
3B08 1246
3B08 1247
3B08 1248
3B08 1249
3B08 1250
3B08 1251
3B08 1252
3B08 1253
3B08 1254
3B08 1255
3B08 1256
3B08 1257
3B08 1258
3B08 1259
3B08 1260
3B08 1261
3B08 1262
3B08 1263
3B08 1264
3B08 1265
3B08 1266
3B08 1267
3B08 1268
3B08 1269
3B08 1270
3B08 1271
3B08 1272
3B08 1273
3B08 1274
3B08 1275
3B08 1276
3B08 1277
3B08 1278
3B08 1279
3B08 1280
3B08 1281
3B08 1282
3B08 1283
3B08 1284
3B08 1285
3B08 1286
3B08 1287
3B08 1288
3B08 1289
3B08 1290
3B08 1291
3B08 1292
3B08 1293
3B08 1294
3B08 1295
3B08 1296
3B08 1297
3B08 1298
3B08 1299
3B08 1300
3B08 1301
3B08 1302
3B08 1303
3B08 1304
3B08 1305
3B08 1306
3B08 1307
3B08 1308
3B08 1309
3B08 1310
3B08 1311
3B08 1312
3B08 1313
3B08 1314
3B08 1315
3B08 1316
3B08 1317
3B08 1318
3B08 1319
3B08 1320
3B08 1321
3B08 1322
3B08 1323
3B08 1324
3B08 1325
3B08 1326
3B08 1327
3B08 1328
3B08 1329
3B08 1330
3B08 1331
3B08 1332
3B08 1333
3B08 1334
3B08 1335
3B08 1336
3B08 1337
3B08 1338
3B08 1339
3B08 1340
3B08 1341
3B08 1342
3B08 1343
3B08 1344
3B08 1345
3B08 1346
3B08 1347
3B08 1348
3B08 1349
3B08 1350
3B08 1351
3B08 1352
3B08 1353
3B08 1354
3B08 1355
3B08 1356
3B08 1357
3B08 1358
3B08 1359
3B08 1360
3B08 1361
3B08 1362
3B08 1363
3B08 1364
3B08 1365
3B08 1366
3B08 1367
3B08 1368
3B08 1369
3B08 1370
3B08 1371
3B08 1372
3B08 1373
3B08 1374
3B08 1375
3B08 1376
3B08 1377
3B08 1378
3B08 1379
3B08 1380
3B08 1381
3B08 1382
3B08 1383
3B08 1384
3B08 1385
3B08 1386
3B08 1387
3B08 1388
3B08 1389
3B08 1390
3B08 1391
3B08 1392
3B08 1393
3B08 1394
3B08 1395
3B08 1396
3B08 1397
3B08 1398
3B08 1399
3B08 1400
3B08 1401
3B08 1402
3B08 1403
3B08 1404
3B08 1405
3B08 1406
3B08 1407
3B08 1408
3B08 1409
3B08 1410
3B08 1411
3B08 1412
3B08 1413
3B08 1414
3B08 1415
3B08 1416
3B08 1417
3B08 1418
3B08 1419
3B08 1420
3B08 1421
3B08 1422
3B08 1423
3B08 1424
3B08 1425
3B08 1426
3B08 1427
3B08 1428
3B08 1429
3B08 1430
3B08 1431
3B08 1432
3B08 1433
3B08 1434
3B08 1435
3B08 1436
3B08 1437
3B08 1438
3B08 1439
3B08 1440
3B08 1441
3B08 1442
3B08 1443
3B08 1444
3B08 1445
3B08 1446
3B08 1447
3B08 1448
3B08 1449
3B08 1450
3B08 1451
3B08 1452
3B08 1453
3B08 1454
3B08 1455
3B08 1456
3B08 1457
3B08 1458
3B08 1459
3B08 1460
3B08 1461
3B08 1462
3B08 1463
3B08 1464
3B08 1465
3B08 1466
3B08 1467
3B08 1468
3B08 1469
3B08 1470
3B08 1471
3B08 1472
3B08 1473
3B08 1474
3B08 1475
3B08 1476
3B08 1477
3B08 1478
3B08 1479
3B08 1480
3B08 1481
3B08 1482
3B08 1483
3B08 1484
3B08 1485
3B08 1486
3B08 1487
3B08 1488
3B08 1489
3B08 1490
3B08 1491
3B08 1492
3B08 1493
3B08 1494
3B08 1495
3B08 1496
3B08 1497
3B08 1498
3B08 1499
3B08 1500
3B08 1501
3B08 1502
3B08 1503
3B08 1504
3B08 1505
3B08 1506
3B08 1507
3B08 1508
3B08 1509
3B08 1510
3B08 1511
3B08 1512
3B08 1513
3B08 1514
3B08 1515
3B08 1516
3B08 1517
3B08 1518
3B08 1519
3B08 1520
3B08 1521
3B08 1522
3B08 1523
3B08 1524
3B08 1525
3B08 1526
3B08 1527
3B08 1528
3B08 1529
3B08 1530
3B08 1531
3B08 1532
3B08 1533
3B08 1534
3B08 1535
3B08 1536
3B08 1537
3B08 1538
3B08 1539
3B08 1540
3B08 1541
3B08 1542
3B08 1543
3B08 1544
3B08 1545
3B08 1546
3B08 1547
3B08 1548
3B08 1549
3B08 1550
3B08 1551
3B08 1552
3B08 1553
3B08 1554
3B08 1555
3B08 1556
3B08 1557
3B08 1558
3B08 1559
3B08 1560
3B08 1561
3B08 1562
3B08 1563
3B08 1564
3B08 1565
3B08 1566
3B08 1567
3B08 1568
3B08 1569
3B08 1570
3B08 157
```


MCS-96 MACRO ASSEMBLER FFT_RUN

```

ERR LOC OBJECT LINE SOURCE STATEMENT
3B08 448
449 LOG_TABLE: ; 16384*10*LOG(n/128) n=128,129,130 ... 256
450
3B08 00002A024F047006 451 DCW 0, 554, 1103, 1648, 2190, 2727, 3260, 3789
3B18 DA10E312E914EA16 452 DCW 4314, 4835, 5353, 5866, 6376, 6883, 7386, 7885
3B28 BD20A92292247826 453 DCW 8381, 8873, 9362, 9848, 10330, 10810, 11286, 11758
3B38 C42F973166333335 454 DCW 12228, 12695, 13158, 13619, 14076, 14531, 14983, 15432
3B48 063EC13F7A413043 455 DCW 15878, 16321, 16762, 17200, 17635, 18067, 18497, 18925
3B58 954B3C4DDF4EB150 456 DCW 19349, 19772, 20191, 20609, 21024, 21436, 21846, 22254
3B68 8458175AA85B365D 457 DCW 22660, 23063, 23464, 23862, 24259, 24653, 25045, 25435
3B78 DE64606E0675D69 458 DCW 25822, 26208, 26592, 26973, 27353, 27730, 28106, 28479
3B88 B370247294730275 459 DCW 28851, 29220, 29588, 29954, 30318, 30680, 31040, 31399
3B98 0B7C687DC7E2F90 460 DCW 31755, 32110, 32463, 32815, 33165, 33512, 33859, 34203
3BA8 F2B647889B8ED8A 461 DCW 34646, 34887, 35227, 35565, 35902, 36236, 36570, 36901
3BB8 7091B892FF934595 462 DCW 37232, 37560, 37887, 38213, 38537, 38860, 39181, 39501
3BC8 8B9BC89C049E3E9F 463 DCW 39819, 40136, 40452, 40765, 41079, 41390, 41700, 42009
3BD8 4CA57EAGAF7DEA8 464 DCW 42316, 42622, 42927, 43230, 43533, 43833, 44133, 44431
3BE8 B9AEE0AF07B12CB2 465 DCW 44729, 45024, 45319, 45612, 45905, 46196, 46486, 46774
3BF8 D6B7F4B811BA2DBB 466 DCW 47062, 47348, 47633, 47917, 48200, 48482, 48763, 49042
3C08 A9C0 467 DCW 49321
3C0A 468
469 LOG_OFFSET: ; 512*10*LOG(2**(15-n)) n= 0,1,2,3 ... 15
470 ; 512*10*LOG(0.5) n= 16,17,18 ... 31
471
3C0A 4F54A4A5445E3F48 472 DCW 23119, 21578, 20037, 18495, 16954, 15413, 13871, 12330
3C1A 252A20241A1E1518 473 DCW 10789, 9248, 7706, 6165, 4624, 3083, 1541, 0
474
3C2A 475 END

```

ASSEMBLY COMPLETED, NO ERROR(S) FOUND.

02/18/86

PAGE 12

MCS-96 MACRO ASSEMBLER FFT_MAIN_APNOTE

02/18/86 PAGE 1

SERIES-III MCS-96 MACRO ASSEMBLER, V1.0

SOURCE FILE: :F2:FTMAIN.A96

OBJECT FILE: :F2:FTMAIN.OBJ

CONTROLS SPECIFIED IN INVOCATION COMMAND: NOSB

```

ERR LOC OBJECT          LINE    SOURCE STATEMENT
1      $pagelength(50)
2
3      FFT_MAIN_APNOTE MODULE MAIN, STACKSIZE(6)
4
5      ; Intel Corporation, January 24, 1986
6      ; by Ira Horden, MCO Applications
7
8
9      ; This program performs an FFT on real data and plots it on a printer.
10     ; It uses the program modules A2DCON, PLOTSP, and FPTRUN. The adjustable
11     ; parameters of each of the programs are set by this main module.
12
13
14     $INCLUDE (:F0:DEMO96.INC) ; Include SFR definitions
15     ;$olist ; Turn listing off for include file
16
17     ;*****
18     ;
19     ; Copyright 1985, Intel Corporation
20     ; October 28, 1985
21     ; by Ira Horden, MCO Applications
22     ;
23     ; DEMO96.INC - DEFINITION OF SYMBOLIC NAMES FOR THE I/O REGISTERS OF THE 8096
24     ;
25     ;*****
26
27     ZERO EQU 00h:WORD ; R/W Zero Register
28     AD_COMMAND EQU 02h:BYTE ; W A to D command register
29     AD_RESULT_LO EQU 02h:BYTE ; R Low byte of result and channel
30     AD_RESULT_HI EQU 03h:BYTE ; R High byte of result
31     HSI_MODE EQU 03h:BYTE ; W Controls HSI transition detector
32     HSO_TIME EQU 04h:WORD ; W HSI time tag
33     HSI_TIME EQU 04h:WORD ; R HSO time tag
34     HSO_COMMAND EQU 05h:BYTE ; W HSO command tag
35     HSI_STATUS EQU 06h:BYTE ; R HSI status register (reads fifo)
36     SBUF EQU 07h:BYTE ; R/W Serial port buffer
37     INT_MASK EQU 08h:BYTE ; R/W Interrupt mask register
38     INT_PENDING EQU 09h:BYTE ; R/W Interrupt pending register
39     SPCON EQU 11h:BYTE ; W Serial port control register
40     SPSTAT EQU 11h:BYTE ; R Serial port status register
41     WATCHDOG EQU 0Ah:BYTE ; W Watchdog timer

```

```

0000
0002
0002
0003
0003
0004
0004
0006
0006
0007
0008
0009
0011
0011
000A

```

270189-45

MCS-96 MACRO ASSEMBLER FFT_MAIN_APNOTE

02/18/86

PAGE 2

```

ERR LOC OBJECT      LINE    SOURCE STATEMENT
000A                -1 42  TIMER1      EQU  0AH:WORD      ; R   Timer1 register
000C                -1 43  TIMER2      EQU  0CH:WORD      ; R   Timer2 register
000E                -1 44  PORT0       EQU  0EH:BYTE      ; R   I/O port 0
000E                -1 45  BAUD_REG    EQU  0EH:BYTE      ; W   Baud rate register
000F                -1 46  PORT1      EQU  0FH:BYTE      ; R/W I/O port 1
0010                -1 47  PORT2      EQU  10H:BYTE      ; R/W I/O port 2
0015                -1 48  IOC0       EQU  15H:BYTE      ; W   I/O control register 0
0015                -1 49  IOS0       EQU  15H:BYTE      ; R   I/O status register 0
0016                -1 50  IOC1       EQU  16H:BYTE      ; W   I/O control register 1
0016                -1 51  IOS1       EQU  16H:BYTE      ; R   I/O status register 1
0017                -1 52  PWM_CONTROL EQU  17H:BYTE      ; W   PWM control register
0018                -1 53  SPREG     EQU  18H:WORD      ; R/W System stack pointer
000D                -1 54  CR         EQU  0DH
000A                -1 55  LF         EQU  0AH
000C                -1 56  PUBLIC ZERO,AD_COMMAND,AD_RESULT_LO,AD_RESULT_HI,HSI_MODE,HSO_TIME,HSI_TIME
000C                -1 57  PUBLIC HSO_COMMAND
000C                -1 58  PUBLIC HSI_STATUS,SBUF,INT_MASK,INT_PENDING,WATCHDOG,TIMER1,TIMER2
000C                -1 59  PUBLIC BAUD_REG,PORT0,PORT1,PORT2,SPSTAT,SPCON,IOC0,IOC1,IOS0,IOS1
000C                -1 60  PUBLIC PWM_CONTROL,SP,CR,LF
001C                -1 61  RSEG at 1CH
001C                -1 62  AX:DSW equ 1 ; Temp registers used in conformance
001E                -1 63  DX:DSW equ 1 ; with PIM-96(tm) conventions.
0020                -1 64  BX:DSW equ 1
0022                -1 65  CX:DSW equ 1
001C                -1 66  AL EQU AX :BYTE
001D                -1 67  AH EQU (AX+1) :BYTE
0020                -1 68  BL EQU BX :BYTE
0020                -1 69  BL EQU BX :BYTE
0020                -1 70  public ax, bx, cx, dx, al, ah, bl
0020                -1 71  $list ; Turn listing back on
0020                -1 72  ; End of include file
0020                -1 73  ; A2D UTILITY COMMANDS/RESPONSES FOR "CONTROL_A2D"
0007                -1 74  busy equ 7
0010                -1 75  con_b0 equ 00010000b;convert to BUFF0
0028                -1 76  dump_b0_p_s equ 00101000b; download BUFF0 as PAIRED SIGNED data
0001                -1 77  AVR_NUM equ 1 ; Number of times to average the waveform
0001                -1 78  ; AVR_NUM < 256

```

270189-46

MCS-96 MACRO ASSEMBLER FFT_MAIN_APNOTE

02/18/86

PAGE 3

```

ERR LOC OBJECT      LINE      SOURCE STATEMENT
                                89
                                90 SCALE_FACTOR equ 0 ; Number of rights shifts performed on
                                91 ; output of FFT. Used to prevent overflow
                                92 ; on summation
                                93
                                94 PLOT_RES equ 256 ; Number of input units per plot unit
                                95 PLOT_RES_2 equ plot_res/2
                                96 PLOT_MAX equ plot_res*145 ; 145 chrs/row
                                97
                                98 PUBLIC scale_factor, plot_res, plot_res_2, plot_max
                                99
0000                                100
0001                                101 OSEG at 24H ; common oseg area
                                102
0024                                103 tmpreal: dsb 1
0028                                104 tmpimag: dsb 1
002C                                105 wndptr: dsb 1
002E                                106 varptr: dsb 1
                                107
0000                                108 RSEG
0000                                109 fft_mode: dsb 1
0001                                110 error: dsb 1
0002                                111 avr_cnt: dsb 1
007C                                112 PUBLIC error, fft_mode
                                113
007C                                114 EXTRN sample_period, control_a2d
                                115
                                116
0080                                117 DSEG at 80h
0080                                118 XREAL: dsb 64 ; For FFT routine
0080                                119 DEST_BUFF_BASE: DSW 64 ; For A2D routine
00C0                                120 XIMAG equ XREAL+64 ; For FFT routine
                                121
                                122 PUBLIC DEST_BUFF_BASE, XREAL, XIMAG
                                123
                                124
0200                                125 DSEG AT 200H
                                126
0200                                127 PLOT_IN: DSW 32 ; For FFT routine
0200                                128 FFT_OUT: DSW 32 ; For FFT routine
0240                                129 BUFF0_BASE: DSW 64 ; For A2D routine
02C0                                130 BUFF1_BASE: DSW 64 ; For A2D routine
                                131
                                132 PUBLIC BUFF0_BASE, BUFF1_BASE, FFT_OUT, PLOT_IN
                                133 $eject
                                134
                                135
                                136
                                137
                                138
                                139
                                140
                                141
                                142
                                143
                                144
                                145
                                146
                                147
                                148
                                149
                                150
                                151
                                152
                                153
                                154
                                155
                                156
                                157
                                158
                                159
                                160
                                161
                                162
                                163
                                164
                                165
                                166
                                167
                                168
                                169
                                170
                                171
                                172
                                173
                                174
                                175
                                176
                                177
                                178
                                179
                                180
                                181
                                182
                                183
                                184
                                185
                                186
                                187
                                188
                                189
                                190
                                191
                                192
                                193
                                194
                                195
                                196
                                197
                                198
                                199

```

310189-48

270189-47

MCS-96 MACRO ASSEMBLER FFT_MAIN_APNOTE

02/18/86

PAGE 4

```

ERR LOC  OBJECT          LINE    SOURCE STATEMENT
2080          134
          135 CSEG AT 2080H
          136
          137 EXTRN  INIT_OUTPUT, DRAW_GRAPH, CON_OUT      ; For Plot Routine
          138 EXTRN  FFT_CALC                                ; For FFT routine
          139 EXTRN  A2D_BUFF_UTIL                        ; For A2D routine
          140
2080 A1000018 R 141 DCM  LD  SP,$STACK
2084 A30100301C 142 DCM  LD  AX,3000H
2089          143 SBE_WAIT:
2089 E01CFD      144 DCM  djnz al,sbe_wait ; WAIT FOR SBE TO CLEAR SERIAL PORT INTERRUPTS
208C E01DFA      145 DCM  djnz ah,sbe_wait
          146
208F EF0000      E 147 BEGIN: CALL INIT_OUTPUT ; Initialize serial port
          148
2092          149 NEW_TRANSFORM_SET:
2092 B10000      R 150 ldb  fft_mode,$0000B ; Bit 0 - Real data / Tabled data#
          151 ; Bit 1 - Windowed / Unwindowed#
          152 ; Bit 2 - 10log Mag^2 / Magnitude#
          153 ; Bit 3 - 256db plot / Normal Plot#
2095 B10102      R 154 ldb  avr_cnt,$avr_num
2098 0120        155 clr  bx
209A C321000200 156 CLRMM: st zero,fft_out[bx] ; clear fft magnitude array
209F 65020020   157 add  bx,$2
20A3 89400020   158 cmp  bx,$64
20A7 DFF1       159 blt  CLRMM
          160
20A9 300004      R 161 C_load: bbc  fft_mode,0,do_tab ; Branch if real data is not used
20AC 2819        162 CALL  LOAD_DATA
20AE 2002        163 br   C_win
          164
20B0 282F        165 do_tab: CALL TABLE_LOAD
          166
20B2 310002      R 167 C_win:  bbc  fft_mode,1,calc ; Branch if windowing is not used
20B5 28CB        168 CALL  DO_WINDOW
          169
20B7 EF0000      E 170 CALC:  CALL FFT_CALC
20BA 980001      R 171 errtrp: cmpb error,zero
20BD D7FB        E 172 jne  errtrp
          173
20BF E00205      R 174 DJNZ  avr_cnt, LOAD_DATA ; repeat for AVR_NUM counts
          175
20C2 EF0000      E 176 CALL  DRAW_GRAPH
          177
20C5 27CB        178 BR   NEW_TRANSFORM_SET
          179 $eject

```

270189-48

MCS-96 MACRO ASSEMBLER FFT_MAIN.APNOTE

02/18/86

PAGE 5

```

ERR LOC OBJECT          LINE    SOURCE STATEMENT
                                180
                                181
                                182
20C7          183      LOAD_DATA:          ;;;;  LOAD DATA INTO RAM
                                184
                                185      ldb      port1,$00          ;**** FOR INDICATION ONLY
20C7 B1000F    E 186      SET_A2D:          ;
20CA          187      ldb      control_a2d,$con_b0      ; Set converter for buffer0
20CA B11000    E 188      orb      control_a2d,$01      ; Convert channel 1
20CD 910100    E 189      id       sample_period,$50      ; 100 us sample period
20D0 A1320000  E 190      CALL     a2d_buff_util      ; Start the conversion process
20D4 EF0000    E 191      jbs      control_a2d,busy,$      ; wait for all conversions to be done
20D7 3F00FD    E 192
                                193      Down_load:
20DA          194      ldb      control_a2d,$dump_b0_p_s      ; download b0 paired/signed
20DA B12800    E 195      CALL     a2d_buff_util
20DD EF0000    E 196      RET
20E0 F0        197
                                198
                                199      TABLE_LOAD:
20E1          200      clr      bx
20E1 0120      201      ld       ax,$DATA0      ; Load tabled data for testing
20E3 A102211C  202      load:      ld       cx,[ax]+
20E7 A21D22    203      ld       dx,[ax]+
20EA A21D1E    204      st       cx,xreal[bx]
20ED C321800022 205      st       dx,ximag[bx]
20F2 C321C0001E 206      add      bx,$2
20F7 65020020  207      cmp      bx,$64
20FB 89400020  208      blt      LOAD
20FF DEE6      209      RET
2101 F0        210
                                211      DATA0:      ; SQUARE WAVE
2102          212
2102 FF/FFF/FFF/FFF/FF 213      DCW      32767, 32767, 32767, 32767, 32767, 32767, 32767, 32767
2112 FF/FFF/FFF/FFF/FF 214      DCW      32767, 32767, 32767, 32767, 32767, 32767, 32767, 32767
2122 FF/FFF/FFF/FFF/FF 215      DCW      32767, 32767, 32767, 32767, 32767, 32767, 32767, 32767
2132 FF/FFF/FFF/FFF/FF 216      DCW      32767, 32767, 32767, 32767, 32767, 32767, 32767, 32767
2142 0180018001800180 217      DCW      -32767, -32767, -32767, -32767, -32767, -32767, -32767, -32767
2152 0180018001800180 218      DCW      -32767, -32767, -32767, -32767, -32767, -32767, -32767, -32767
2162 0180018001800180 219      DCW      -32767, -32767, -32767, -32767, -32767, -32767, -32767, -32767
2172 0180018001800180 220      DCW      -32767, -32767, -32767, -32767, -32767, -32767, -32767, -32767
221
222      $eject

```

270189-49

ERR	LOC	OBJECT	LINE	SOURCE STATEMENT
			255	
	3D00		256	CSEGT AT 3D00H ; ADDITIONAL TABLES FOR TESTING
			257	; SINE 7.0 X
	3D00		258	DATA1:
	3D00	00003351897DE270	259	DCW 0, 20787, 32137, 28898, 12539, -9512, -27245, -32609
	3D10	7EA574F31C477C7A	260	DCW -23170, -3212, 18204, 31356, 30273, 15446, -6393, -25329
	3D20	01800F9D08E7563C	261	DCW -32767, -25329, -6392, 15446, 30273, 31356, 18204, -3212
	3D30	7EA59F809395D8DA	262	DCW -23170, -32609, -27245, -9512, 12539, 28898, 32137, 20787
	3D40	0000CDAE77821E8F	263	DCW -0, -20787, -32137, -28898, -12539, 9512, 27245, 32609
	3D50	825A8C0C84B88485	264	DCW 23170, 3212, -18204, -31356, -30273, -15446, 6393, 25329
	3D60	FF7FF162F818AAC3	265	DCW 32767, 25329, 6392, -15446, -30273, -31356, -18204, 3212
	3D70	825A617F6D6A2825	266	DCW 23170, 32609, 27245, 9512, -12539, -28898, -32137, -20787
			267	
	3D80		268	DATA2: ; SINE 7.5 X
			269	
	3D80	0000F555617FCF66	270	DCW 0, 22005, 32609, 26319, 6393, -16846, -31356, -29621
	3D90	05CF1F2BE270297C	271	DCW -12539, 11039, 28898, 31785, 18204, -4808, -25329, -32728
	3DA0	7EA58BF933519C7E	272	DCW -23170, -1608, 20787, 32412, 27245, 7962, -15446, -30852
	3DB0	BF8946C92825C96D	273	DCW -30273, -14010, 9512, 28105, 32137, 19519, -3212, -24279
	3DC0	018029A174F33F4C	274	DCW -32767, -24279, -3212, 19519, 32137, 28105, 9512, -14010
	3DD0	BF897C87AAC31A1F	275	DCW -30273, -30852, -15446, 7962, 27245, 32412, 20787, -1608
	3DE0	7EA528800F9D38ED	276	DCW -23170, -32728, -25329, -4808, 18205, 31785, 28898, 11039
	3DF0	05CF4B8C848533BE	277	DCW -12539, -29621, -31356, -16846, 6393, 26319, 32609, 22005
			278	
	3E00		279	DATA3: ; .707*SINE 7.5X
			280	
	3E00	0000C630F5AAF48	281	DCW 0, 15558, 23055, 18607, 4520, -11910, -22169, -20942
	3E10	5FDD7C1ECF4FC857	282	DCW -8865, 7804, 20431, 22472, 12870, -3399, -17908, -23138
	3E20	03C08FFB69398459	283	DCW -16381, -1137, 14697, 22916, 19262, 5629, -10921, -21812
	3E30	65AC4FD9451A9E4D	284	DCW -21403, -9905, 6725, 19870, 22721, 13800, -2271, -17165
	3E40	82A5F3BC21F7E835	285	DCW -23166, -17165, -2271, 13800, 22721, 19870, 6725, -9905
	3E50	65ACCCAA58D5FD15	286	DCW -21403, -21812, -10920, 5629, 19262, 22916, 14696, -1137
	3E60	03C09EA50CBAB9F2	287	DCW -16381, -23138, -17908, -3399, 12871, 22472, 20431, 7804
	3E70	5FDD32AE67A97AD1	288	DCW -8865, -20942, -22169, -11910, 4520, 18607, 23055, 15557
			289	
	3E80		290	DATA4: ; .707*SINE(11X) /16
			291	
	3E80	0000FD04B40472FF	292	DCW 0, 1277, 1204, -142, -1338, -1119, 282, 1386
	3E90	00045CFE74FA69FC	293	DCW 1024, -420, -1420, -919, 554, 1441, 804, -683
	3EA0	58FA55FD2403A105	294	DCW -1448, -683, 804, 1441, 554, -919, -1420, -420
	3EB0	00046A051A01A1FB	295	DCW 1024, 1386, 282, -1119, -1338, -142, 1204, 1277
	3EC0	000003FB4CFB8E00	296	DCW -0, -1277, -1204, 142, 1338, 1119, -282, -1386
	3ED0	00FCA4018C059703	297	DCW -1024, 420, 1420, 919, -554, -1441, -804, 683
	3EE0	A805AB02DCFC5FFA	298	DCW 1448, 683, -804, -1441, -554, 919, 1420, 420
	3EF0	00FC96FAR6FE5F04	299	DCW -1024, -1386, -282, 1119, 1338, 142, -1204, -1277
			300	
	3F00		301	DATA5: ; .707*(SINE 7.5X + 1/16 SINE 11X)

SL0189-20

270189-51

MCS-96 MACRO ASSEMBLER FFT_MAIN_APNOTE

02/18/86

PAGE 8

ERR LOC	OBJECT	LINE	SOURCE STATEMENT
		302	
3F00	0000C241C35E2148	303	DCW 0, 16834, 24259, 18465, 3182, -13029, -21886, -19557
3F10	5EE1D81C43A3154	304	DCW -7842, 7384, 19011, 21553, 13425, -1958, -17103, -23821
3F20	5BBAE5F8ED3C245F	305	DCW -17829, -1819, 15501, 24356, 19816, 4710, -12341, -22232
3F30	65B0B9DE5F1B3F49	306	DCW -20379, -8519, 7007, 18751, 21383, 13658, -1067, -15888
3F40	82A5F6B76DF27636	307	DCW -23166, -18442, -3475, 13942, 24059, 20990, 6442, -11290
3F50	65A870AC8DA9419	308	DCW -22427, -21392, -9500, 6548, 18708, 21475, 13892, -454
3F60	ABC548A8EB618ED	309	DCW -14933, -22456, -18712, -4840, 12317, 23391, 21851, 8225
3F70	5FD9C8A84DA8D9D5	310	DCW -9889, -22328, -22451, -10791, 5857, 18749, 21851, 14281
		311	
		312	
3F80		313	END

ASSEMBLY COMPLETED, NO ERROR(S) FOUND.

270189-52

SERIES-III MCS-96 RELOCATOR AND LINKER, V2.0
Copyright 1983 Intel Corporation

INPUT FILES: :F2:FTMAIN.OBJ, :F2:FFTRUN.OBJ, :F2:PLOTSP.OBJ, :F2:A2DCON.OBJ

OUTPUT FILE: :F2:FFTOUT

CONTROLS SPECIFIED IN INVOCATION COMMAND:

IX

INPUT MODULES INCLUDED:

:F2:FTMAIN.OBJ(FFT_MAIN_APNOTE) 02/18/86

:F2:FFTRUN.OBJ(FFT_RUN) 02/18/86

:F2:PLOTSP.OBJ(PLOT_SERIAL) 02/18/86

:F2:A2DCON.OBJ(A2D_BUFFERING_UTILITY) 02/18/86

SEGMENT MAP FOR :F2:FFTOUT(FFT_MAIN_APNOTE):

	TYPE	BASE	LENGTH	ALIGNMENT	MODULE NAME
**RESERVED*		0000H	001AH		
	REG	001AH	0001H	BYTE	PLOT_SERIAL
*** GAP ***		001BH	0001H		
	REG	001CH	0008H	ABSOLUTE	FFT_MAIN_APNOTE
	OVRLY	0024H	0035H	ABSOLUTE	FFT_RUN
OVERLAP	OVRLY	0024H	0010H	ABSOLUTE	PLOT_SERIAL
OVERLAP	OVRLY	0024H	000CH	ABSOLUTE	FFT_MAIN_APNOTE
*** GAP ***		0059H	0001H		
	OVRLY	005AH	0006H	WORD	A2D_BUFFERING_UTILITY
	REG	0060H	000CH	WORD	A2D_BUFFERING_UTILITY
	REG	006CH	0003H	BYTE	FFT_MAIN_APNOTE
*** GAP ***		006FH	0011H		
	DATA	0080H	0080H	ABSOLUTE	FFT_MAIN_APNOTE
	STACK	0100H	001EH	WORD	
	DATA	011EH	0080H	WORD	FFT_RUN
*** GAP ***		019EH	0062H		
	DATA	0200H	0140H	ABSOLUTE	FFT_MAIN_APNOTE
*** GAP ***		0340H	1CC2H		
	CODE	2002H	0002H	ABSOLUTE	A2D_BUFFERING_UTILITY
*** GAP ***		2004H	007CH		
	CODE	2080H	01C0H	ABSOLUTE	FFT_MAIN_APNOTE
*** GAP ***		2240H	0040H		
	CODE	2280H	0215H	ABSOLUTE	FFT_RUN
*** GAP ***		2495H	006BH		
	CODE	2500H	0168H	ABSOLUTE	PLOT_SERIAL
	CODE	2688H	00ECH	BYTE	A2D_BUFFERING_UTILITY
*** GAP ***		2754H	10ACH		
	CODE	3800H	042AH	ABSOLUTE	FFT_RUN
*** GAP ***		3C2AH	00D6H		
	CODE	3D00H	0280H	ABSOLUTE	FFT_MAIN_APNOTE
*** GAP ***		3F80H	C080H		

270189-53

Listing 3—Main Routine (Continued)

Several constants are then setup for other routines. The purpose of centrally locating these constants was the ease of modifying the operation of the routines. Note that `AVR_NUM` and `SCALE_FACTOR` must be changed at the same time. `SCALE_FACTOR` is the shift count used to divide each FFT output value before it is added to the output array. `AVR_NUM` must be less than $2^{**SCALE_FACTOR}$ or an overflow could occur. Next, the public variables are declared for the arrays and a few other parameters.

The program then begins by setting the stack pointer and waiting for the SBE-96 to finish talking to the terminal. If this is not done, there may be serial port interrupts occurring for the first twenty five milliseconds of program operation.

Initialization of the plotter is next, followed by setting the `FFT_MODE` byte. This byte controls the graphing, loading and magnitude calculation of the FFT data. Since `FFT_MODE` is declared `PUBLIC` in this module, and `EXTERNAL` in the `PLOT` module and `FFTRUN` module, the extra bits available in this byte can be used for future enhancements.

The next step is to clear the FFT output array. Since the FFT program can be set to average its results by dividing the output before adding it to the magnitude array, the array must be cleared before beginning the program.

Data is then loaded into the FFT input array by the code at `LOAD_DATA`, or the code at `TABLE_LOAD`, depending on the value of `FFT_MODE` bit 0. The tabled data located at `DATA0` is a square wave of magnitude 1. This waveform provides a reasonable test of the FFT algorithm, as many harmonics are generated. The results are also easy to check as the pattern contains half zeros, imaginary values which are always the same, and real values which decrease. Figure 13 shows the output in fractions, hexadecimal and decimal. The hexadecimal and decimal values are based on an output of 16384 being equal to 1.00.

Note that the magnitude is

$$SQRT(REAL^2 + IMAG^2)$$

and the dB value is

$$10 \text{ LOG } ((REAL^2 + IMAG^2)/65536)$$

The divide by 65536 is used for the dB scale to provide a reasonable range for calculations. If this was not done, a 32-bit LOG function would have been needed.

After the data is loaded, the data is optionally windowed, based on `FFT_MODE` bit 1, and the FFT program is called. Once the loop has been performed `AVR_CNT` times, the graph is drawn by the plot routine.

Appended to the main routine is the `FFTOUT.M96` Listing. This is provided by the relocater and linker, `RL96`. With this listing and the main program, it is possible to determine which sections of code are at which addresses.

Using the modular programming methods employed here, it is reasonably easy to debug code. By emulating the program in a relatively high level language, each routine can be checked for functionality against a known standard. The closer the high level implementation matches the `ASM96` version, the more possible checkpoints there are between the two routines.

Once all of the program routines (modules) can be shown to work individually, the main program should work unless there is unwanted interaction between the modules. These interactions can be checked by verifying the inputs and outputs of each module. The assembly language locations to perform the program breaks can be retrieved by absolutely locating the main module. The other modules can be dynamically located by `RL96`.

The more interactive program modules are, the more difficult the program becomes to debug. This is especially true when multiple interrupts are occurring, and several of the interrupt routines are themselves interruptable. In these cases, it may be necessary to use debugging equipment with trace capability, like the `VLSICE-96`. If this type of equipment is not available, then using I/O ports to indicate the entering and leaving of each routine may be useful. In this way it will be possible to watch the action of the program on an oscilloscope or logic analyzer. There are several places within this code that I/O port toggling has been used as an aid to debugging the program. These lines of code are marked "FOR INDICATION ONLY."

K	Fractional			dB	Decimal			Hexadecimal		
	REAL	IMAG	MAG ²		REAL	IMAG	MAG ²	REAL	IMAG	MAG ²
0	0.0000	0.0000	0.0000	0.000	0	0	0	0	0	0
1	0.0625	-1.2722	1.2738	38.225	1024	-20843	20868	400	AE95	5184
2	0.0000	0.0000	0.0000	0.000	0	0	0	0	0	0
3	0.0625	-0.4213	0.4260	28.710	1024	-6903	6978	400	E509	1B42
4	0.0000	0.0000	0.0000	0.000	0	0	0	0	0	0
5	0.0625	-0.2495	0.2572	24.329	1024	-4088	4214	400	F008	1076
6	0.0000	0.0000	0.0000	0.000	0	0	0	0	0	0
7	0.0625	-0.1747	0.1855	21.491	1024	-2862	3039	400	F4D2	BDF
8	0.0000	0.0000	0.0000	0.000	0	0	0	0	0	0
9	0.0625	-0.1321	0.1462	19.421	1024	-2165	2395	400	F78B	95B
10	0.0000	0.0000	0.0000	0.000	0	0	0	0	0	0
11	0.0625	-0.1043	0.1216	17.820	1024	-1708	1992	400	F954	7C8
12	0.0000	0.0000	0.0000	0.000	0	0	0	0	0	0
13	0.0625	-0.0843	0.1049	16.540	1024	-1381	1719	400	FA9B	6B7
14	0.0000	0.0000	0.0000	0.000	0	0	0	0	0	0
15	0.0625	-0.0690	0.0931	15.499	1024	-1130	1525	400	FB96	5F5
16	0.0000	0.0000	0.0000	0.000	0	0	0	0	0	0
17	0.0625	-0.0566	0.0844	14.645	1024	-928	1382	400	FC60	566
18	0.0000	0.0000	0.0000	0.000	0	0	0	0	0	0
19	0.0625	-0.0464	0.0778	13.944	1024	-759	1275	400	FD09	4FB
20	0.0000	0.0000	0.0000	0.000	0	0	0	0	0	0
21	0.0625	-0.0375	0.0729	13.374	1024	-614	1194	400	FD9A	4AA
22	0.0000	0.0000	0.0000	0.000	0	0	0	0	0	0
23	0.0625	-0.0296	0.0691	12.918	1024	-484	1133	400	FE1C	46D
24	0.0000	0.0000	0.0000	0.000	0	0	0	0	0	0
25	0.0625	-0.0224	0.0664	12.564	1024	-366	1088	400	FE92	440
26	0.0000	0.0000	0.0000	0.000	0	0	0	0	0	0
27	0.0625	-0.0157	0.0644	12.305	1024	-256	1056	400	FF00	420
28	0.0000	0.0000	0.0000	0.000	0	0	0	0	0	0
29	0.0625	-0.0093	0.0632	12.135	1024	-152	1035	400	FF68	40B
30	0.0000	0.0000	0.0000	0.000	0	0	0	0	0	0
31	0.0625	-0.0031	0.0626	12.051	1024	-50	1025	400	FFCE	401

Figure 13. FFT Output for a Square Wave Input

9.0 ANALOG TO DIGITAL CONVERTER MODULE

The module presented in Listing 4 is a general purpose one which converts analog values under interrupt control and stores them in one of two buffers. These buffers

can then be downloaded to another buffer, such as the input buffer to the FFT program. During downloading, this module can convert the data into signed or unsigned formats, and fill a linear or a paired array. A paired array is like the one used in the FFT transform program. It requires N data points placed alternately in two arrays, one starting at zero and the other at N/2.

MCS-96 MACRO ASSEMBLER A2D_BUFFERING_UTILITY

02/18/86

PAGE 1

SERIES-III MCS-96 MACRO ASSEMBLER, V1.0

SOURCE FILE: :F2:A2DCON.A96

OBJECT FILE: :F2:A2DCON.OBJ

CONTROLS SPECIFIED IN INVOCATION COMMAND: NOSB

```

ERR LOC OBJECT          LINE    SOURCE STATEMENT
1          $pagelength(50)
2
3      A2D_Buffering_Utility  module stacksize(12)
4      ;
5      ; Intel Corporation, July 16, 1985
6      ; by Dave Ryan, Intel Applications Engineer
7      ;
8      ; This utility fills a memory buffer with A/D conversion results. The
9      ; conversions are done under interrupt control, and are initiated when
10     ; A2D_BUFF_Util is called. The results of the conversions are placed
11     ; in one of two buffers, called BUFF0 and BUFF1.
12     ;
13     ; This utility provides options for the selection of the buffer lengths, data
14     ; format, sample period, conversion channel and time base. The utility also
15     ; has a download routine that will load either buffer into a register file
16     ; buffer. Output formats can also be chosen for the downloaded buffer. The
17     ; data can be formatted as signed or unsigned linear or paired arrays.
18     ;
19     ; RUN-TIME OPTIONS
20     ;
21     ; Rather than use the STACK to pass controls, this utility gets its directions
22     ; from 2 control words in memory. The utility expects that its control words
23     ; are valid at the time A2D_BUFF_Util is called and remain valid throughout
24     ; A/D interrupt executions and downloads. The control words are:
25     ;
26     ; Sample_Period ; WORD ; The time between samples in timer counts
27     ; ; where the timer used has been specified
28     ;
29     ;
30     ; Control_A2D ; BYTE ; Control information for the utility:
31     ; BIT#
32     ;
33     ; 0-2 ; Channel Number
34     ; 3 ; Signed Result/Unsigned Result#
35     ; 4 ; Convert/Download#
36     ; 5 ; BUFF1/BUFF0# for conversions
37     ; BUFF0/BUFF1# for downloads
38     ; 6 ; Linear/Paired#
39     ; 7 ; Converter BUSY/IDLE#
40     ;
41     $EJECT

```

S30188-28

270189-54

MCS-86 MACRO ASSEMBLER A2D_BUFFERING_UTILITY

02/18/86

PAGE 2

ERR LOC OBJECT

LINE

SOURCE STATEMENT

```

42 ;
43 ; The following is a table of equates that can be used to simplify the
44 ; bit diddling requirements. If you are not running conversions concurrently
45 ; with downloads, always LDB Control_A2D with the following command then
46 ; ORB Control_A2D with the channel number you wish to convert if you are
47 ; starting a conversion.
48 ;
49 ; Once the utility is called, care must be taken when Control_A2d is
50 ; modified. You can cause downloads to occur while conversions are running,
51 ; but you cannot start conversions during a download. To do this, ORB to the
52 ; control byte with the appropriate bits set. Do NOT change the BUFF bit or
53 ; the BUSY bit. Just set the download bit and set the data format bits to the
54 ; correct values.
55 ;
56 ; The BUFF bit has opposite definitions for conversions and downloads. This
57 ; allows conversions to be done into BUFF0 while downloads come from BUFF1, and
58 ; vice versa.
59 ;
60 ; A2D UTILITY COMMANDS
61 ;
62 ;con_b0 equ 00010000b; convert to BUFF0
63 ;con_b1 equ 00110000b; " BUFF1
64 ;
65 ;dump_b0_l_u equ 01100000b; download BUFF0 as LINEAR UNSIGNED data
66 ;dump_b1_l_u equ 01000000b; " " " "
67 ;dump_b0_p_u equ 00100000b; " " " PAIRED "
68 ;dump_b1_p_u equ 00000000b; " " " " "
69 ;dump_b0_l_s equ 01101000b; download BUFF0 as LINEAR SIGNED data
70 ;dump_b1_l_s equ 01001000b; " " " "
71 ;dump_b0_p_s equ 00101000b; " " " PAIRED "
72 ;dump_b1_p_s equ 00001000b; " " " " "
73 ;
74 $eject

```

270189-55

ERR LOC OBJECT

LINE

SOURCE STATEMENT

CONVERSIONS SPECIFIED IN TRANSLATION COMMANDS: NONE

OBJECT FILE: A2D_BUFFERING_UTILITY

SOURCE FILE: A2D_BUFFERING_UTILITY

SERIES-III MCS-86 MICRO CONVERTER AT 5

MCS-86 MICRO CONVERTER V2D BUFFERING_UTILITY

02/18/86

PAGE 2

MCS-96 MACRO ASSEMBLER A2D_BUFFERING_UTILITY

02/18/86

PAGE 3

```

ERR LOC OBJECT      LINE      SOURCE STATEMENT
75 ;
76 ; ASSEMBLY-TIME OPTIONS
77 ;
78 ; The base addresses and length of each conversion buffer and the destination
79 ; buffer are DECLARED EXTERNAL in this utility. Other options such as selection
80 ; of the timer used as a timebase, the length of the buffer, and the effective
81 ; number of bits in the reported result are set at assembly time through use
82 ; of EQUates in this module.
83 ;
84 ; The following parameters need to be provided at assembly or link time.
85 ; The buffer bases are declared EXTERNAL by this utility, while the buffer
86 ; length shift count and HSO commands are EQUated.
87 ;
88 ; BUFF0_BASE      ; The starting address of BUFF0
89 ; BUFF1_BASE      ; The starting address of BUFF1
90 ; DEST_BUFF_BASE  ; The starting address of the download
91 ;                  ; target buffer.
92 ;
93 ; BUFF_LENGTH     ; The number of SAMPLES that each
94 ;                  ; buffer must hold. must be >1 and <256
95 ;
96 ; Shift_count     ; The number of times that the conversion result is
97 ;                  ; to be shifted right from its natural left justified
98 ;                  ; position. Setting a shift count greater than 6 will
99 ;                  ; result in lost bits to the right. Rounding is NOT
100 ;                  ; done.
101 ;
102 ; CLOCK           ; Specify as either TIMER1 or T2CLK. This is the
103 ;                  ; timebase used for conversions.
104 ;
105 ; Samples are stored as words in the buffers. The program stores
106 ; conversions linearly in BUFF0 and BUFF1, and linearly or paired in the
107 ; destination buffer as selected. If the download is to be paired, the first
108 ; sample is placed in location DEST_BUFF_BASE, the second sample is placed in
109 ; location (DEST_BUFF_BASE + BUFF_LENGTH), the third in (DEST_BUFF_BASE + 2),
110 ; the fourth in (DEST_BUFF_BASE + 2 + BUFF_LENGTH), etc.
111 ;
112 $object

```

530100-01

270189-56

MCS-96 MACRO ASSEMBLER

FILE

SOURCE STATEMENT

MCS-96 MACRO ASSEMBLER

A2D_BUFFERING_UTILITY

02/18/86

PAGE 3

ERR LOC	OBJECT	LINE	SOURCE STATEMENT
		113	;
		114	;NOTES ON EXECUTION
		115	;
		116	; When a utility call directs the initiation of a set of A2D conversions, the
		117	; first conversion is begun at approximately one sample time plus 50 state
		118	; times from when the utility was called. This assumes that no interrupts are
		119	; present.
		120	;
		121	; The conversion busy bit is set approximately 50 state times after a call
		122	; to the utility, if the convert bit was set in the A2D_Control byte. The
		123	; busy bit is cleared after all conversion results have been stored in the
		124	; result buffer designated (BUFF0 or BUFF1).
		125	;
		126	; Take great care in modifying the A2D_Control byte to do a download while
		127	; conversions are taking place. You can never download a buffer that is
		128	; being converted into. The results would be invalid.
		129	\$eject

270189-57

Listing 4—A to D Converter Routine (Continued)

6-146

MCS-96 MACRO ASSEMBLER A2D_BUFFERING_UTILITY

02/18/86

PAGE 5

```

ERR LOC OBJECT      LINE      SOURCE STATEMENT
                                130
                                131      RSEG
                                132
                                133      EXTRN BUFF0_BASE, BUFF1_BASE, DEST_BUFF_BASE
                                134      EXTRN ad_command, ad_result_lo, ad_result_hi
                                135      EXTRN hso_command, hso_time, sp
                                136
                                137      BUFF_LENGTH      EQU      64
                                138      Shift_Count      EQU      1
                                139      CLOCK              EQU      TIMER1
                                140
                                141      ; set up hso commands for correct timer *****
                                142
                                143      TIMER1              equ      0AH
                                144      T2CLK              equ      0CH
                                145
                                146      MASK              equ      (10h*CLOCK)AND(40h)
                                147
                                148      Start_A2D          equ      (00001111b)OR(MASK)
                                149      ;start a2d based on timer 1, no interrupt
                                150
                                151      HSO_0_Low          equ      (00000000b)OR(MASK)
                                152      ; make hso.0 low based on timer1 no interrupt
                                153
                                154      HSO_0_High         equ      (00100000b)OR(MASK)
                                155      ; make hso.0 hi based on timer1 no interrupt
                                156
                                157
                                158      ; set up storage *****
                                159
                                160      adudtemp0:         DSW      1; temp register for download calls
                                161
                                162      aductemp0:         DSW      1; temp registers for conversion calls
                                163      aductemp1:         DSW      1
                                164      top_of_buffer:     DSW      1
                                165      sample_count:     DSB      1
                                166
                                167      Control_A2D:        DSB      1; the byte that controls the utility execution
                                168      DForm              equ      3      ; Signed/Unsigned#
                                169      Con_Dwn             equ      4      ; Convert/Download#
                                170      B0_B1              equ      5      ; Buffl/Buff0# for conversions
                                171      ; Buff0/Buff1# for downloads
                                172      Lin_Par            equ      6      ; Linear/Paired#
                                173      Busy              equ      10000000B ; Bit 8
                                174      $reject

```

270189-58

PAGR 6

270189-59

Listing 4—A to D Converter Routine (Continued)

6-148

MCS-96 MACRO ASSEMBLER A2D_BUFFERING_UTILITY 02/18/86 PAGE 7

ERR	LOC	OBJECT	LINE	SOURCE STATEMENT
			207	
	0000		208	A2D_BUFF_Util:
			209	
	0000 3C0962	R	210	JBS Control_A2D, Con_Dwn, Convert ; Select convert or download
	0003		211	Download:
	0003 A1000000	E	212	LD src_ptr,#BUFF1_BASE
	0007 350904	R	213	JBC Control_A2D, B0_B1, Set_Data_Format
			214	
	000A		215	Download_BUFF0:
	000A A1000000	E	216	LD src_ptr,#BUFF0_BASE
			217	
	0011		218	
	000E 330900	R	219	Set_Data_Format: ; Choose linear or paired
	000E A1000002	E	220	LD dest_ptr,#DEST_BUFF_BASE
	0012 B14004	R	221	LDB loop_count,#BUFF_LENGTH
	0015 3B091D	R	222	JBS Control_A2D, Lin_Par, Linear_data_loop
			223	
			224	
	0018 180104	R	225	PAIRED: SHRB loop_count,#1 ; The paired data routine uses 1/2
			226	; as many loops as the unpaired
	001B		227	Paired_Data_loop:
	001B A20000	R	228	LD adudtemp0,[src_ptr]+ ; Move even word
	001E C20200	R	229	ST adudtemp0,[dest_ptr]
	0021 65400002	R	230	ADD dest_ptr,#BUFF_LENGTH ; Length = # of words = 1/2 # of bytes
			231	
	0025 A20000	R	232	LD adudtemp0,[src_ptr]+ ; Move odd word
	0028 C20200	R	233	ST adudtemp0,[dest_ptr]+
	002B 69400002	R	234	SUB dest_ptr,#BUFF_LENGTH
			235	
	002F E004E9	R	236	DJNZ loop_count, Paired_Data_loop ; Loop until done
			237	
	0032 280D	E	238	CALL Convert_Data
	0034 F0		239	RET
			240	
			241	
	0035		242	Linear_Data_loop: ; Move data linearly
	0035 A20000	R	243	LD adudtemp0,[src_ptr]+
	0038 C20200	R	244	ST adudtemp0,[dest_ptr]+
			245	
	003B E004F7	R	246	DJNZ loop_count, Linear_Data_loop ; Loop until done
			247	
	003E 2801	E	248	CALL Convert_Data
	0040 F0		249	RET
			250	\$ject

270189-60

MCS-96 MACRO ASSEMBLER A2D_BUFFERING_UTILITY 02/18/86 PAGE 8

ERR	LOC	OBJECT	LINE	SOURCE STATEMENT
			251	
	0041		252	Convert_Data: ; Convert the data in the destination buffer
			253	
	0041 A1400004	R	254	LD loop_count,#BUFF_LENGTH
	0045 A1000000	E	255	LD src_ptr,#DEST_BUFF_BASE
			256	
	0049 A20000	R	257	Again: LD adudtemp0,[src_ptr]
	004C 71C000	R	258	ANDB adudtemp0,#11000000b
	004F 330909	R	259	JBC Control_A2D, DForm, Unsigned_Result
			260	
	0052 000000	E	261	Signed_Result: ;
	0052 69K07F00	R	262	SUB adudtemp0,#7fe0H
	0056 0A0100	R	263	SHRA adudtemp0,#Shift_Count
	0059 2003	R	264	BR Replace_Sample
			265	
	005B 00	E	266	Unsigned_Result: ;
	005B 080100	R	267	SHR adudtemp0,#Shift_Count
			268	
	005E 000000	E	269	Replace_Sample: ;
	005E C20000	R	270	ST adudtemp0,[src_ptr]+
	0061 E004E5	R	271	DJNZ loop_count,Again ; Loop until done
			272	
	0064 F0 000	E	273	RET
			274	
	0065 000000	E	275	Convert: ;
	0065 000000	E	276	;; Prepare to Start Conversions
	0065 F2	R	277	PUSHF
			278	
	0066 918009	R	279	ORB Control_A2D,#Busy ; set converter busy bit
			280	
	0069 B13F08	R	281	LDB sample_count,#BUFF_LENGTH - 1
	006C A1000006	E	282	LD top_of_buffer,#BUFF0_BASE
	0070 A1800004	E	283	LD aductemp1,(BUFF0_BASE + 2*BUFF_LENGTH)
			284	
	0074 350908	R	285	JBC Control_A2D, B0 B1, Start Conversions
	0077 A1000006	E	286	LD top_of_buffer,#BUFF1_BASE
	007B A1800004	E	287	LD aductemp1,(BUFF1_BASE + 2*BUFF_LENGTH)
			288	
	007F 000000	E	289	\$select
			290	
	0081 000000	E	291	
	0081 000000	E	292	
	0083 000000	E	293	
	0085 000000	E	294	
	0087 000000	E	295	
	0089 000000	E	296	
	0091 000000	E	297	
	0093 000000	E	298	
	0095 000000	E	299	
	0097 000000	E	300	
	0099 000000	E	301	
	009B 000000	E	302	
	009D 000000	E	303	
	009F 000000	E	304	
	00A1 000000	E	305	
	00A3 000000	E	306	
	00A5 000000	E	307	
	00A7 000000	E	308	
	00A9 000000	E	309	
	00AB 000000	E	310	
	00AD 000000	E	311	
	00AF 000000	E	312	
	00B1 000000	E	313	
	00B3 000000	E	314	
	00B5 000000	E	315	
	00B7 000000	E	316	
	00B9 000000	E	317	
	00BB 000000	E	318	
	00BD 000000	E	319	
	00BF 000000	E	320	
	00C1 000000	E	321	
	00C3 000000	E	322	
	00C5 000000	E	323	
	00C7 000000	E	324	
	00C9 000000	E	325	
	00CB 000000	E	326	
	00CD 000000	E	327	
	00CF 000000	E	328	
	00D1 000000	E	329	
	00D3 000000	E	330	
	00D5 000000	E	331	
	00D7 000000	E	332	
	00D9 000000	E	333	
	00DB 000000	E	334	
	00DD 000000	E	335	
	00DF 000000	E	336	
	00E1 000000	E	337	
	00E3 000000	E	338	
	00E5 000000	E	339	
	00E7 000000	E	340	
	00E9 000000	E	341	
	00EB 000000	E	342	
	00ED 000000	E	343	
	00EF 000000	E	344	
	00F1 000000	E	345	
	00F3 000000	E	346	
	00F5 000000	E	347	
	00F7 000000	E	348	
	00F9 000000	E	349	
	00FB 000000	E	350	
	00FD 000000	E	351	
	00FF 000000	E	352	

270189-61

MCS-96 MACRO ASSEMBLER A2D_BUFFERING_UTILITY

02/18/86 PAGE 9

```

ERR LOC OBJECT LINE SOURCE STATEMENT
007F 51070900 R 293 ANDB ad_command,Control_A2D,#00000111b ;load channel number
0083 440A0A02 R 295 ADD aductemp0,CLOCK,Sample_Period ;start first conversion
                                296 ;one sample time from
                                297 ;now
                                298
                                299 Load_HSO_Command Start_A2D ; Start A2D at Time=aductemp0
008D CC00 R 304 POP temp ; get a copy of the psw
                                305
                                306 Load_HSO_Command HSO_0_high ; set hao.0 high at conversion
                                310 ; start time for external S/H
                                311
0095 81020200 R 312 OR temp,#202h ; enable a2d interrupts
                                313
0099 640A02 R 314 ADD aductemp0,Sample_Period
                                315
                                316 Load_HSO_Command Start_A2D ; start second conversion one
                                320 ; sample time from the first
                                321
00A2 C800 R 322 PUSH temp ; put psw back on stack
                                323
                                324 Load_HSO_Command HSO_0_low ;lower hao.0 for external S/H
                                325
                                326
                                327 POPF
                                328
                                329 RET
                                330
                                331 $eject
                                332
                                333
                                334
                                335
                                336
                                337
                                338
                                339
                                340
                                341
                                342
                                343
                                344
                                345
                                346
                                347
                                348
                                349
                                350
                                351
                                352
                                353
                                354
                                355
                                356
                                357
                                358
                                359
                                360
                                361
                                362
                                363
                                364
                                365
                                366
                                367
                                368
                                369
                                370
                                371
                                372
                                373
                                374
                                375
                                376
                                377
                                378
                                379
                                380
                                381
                                382
                                383
                                384
                                385
                                386
                                387
                                388
                                389
                                390
                                391
                                392
                                393
                                394
                                395
                                396
                                397
                                398
                                399
                                400
                                401
                                402
                                403
                                404
                                405
                                406
                                407
                                408
                                409
                                410
                                411
                                412
                                413
                                414
                                415
                                416
                                417
                                418
                                419
                                420
                                421
                                422
                                423
                                424
                                425
                                426
                                427
                                428
                                429
                                430
                                431
                                432
                                433
                                434
                                435
                                436
                                437
                                438
                                439
                                440
                                441
                                442
                                443
                                444
                                445
                                446
                                447
                                448
                                449
                                450
                                451
                                452
                                453
                                454
                                455
                                456
                                457
                                458
                                459
                                460
                                461
                                462
                                463
                                464
                                465
                                466
                                467
                                468
                                469
                                470
                                471
                                472
                                473
                                474
                                475
                                476
                                477
                                478
                                479
                                480
                                481
                                482
                                483
                                484
                                485
                                486
                                487
                                488
                                489
                                490
                                491
                                492
                                493
                                494
                                495
                                496
                                497
                                498
                                499
                                500
                                501
                                502
                                503
                                504
                                505
                                506
                                507
                                508
                                509
                                510
                                511
                                512
                                513
                                514
                                515
                                516
                                517
                                518
                                519
                                520
                                521
                                522
                                523
                                524
                                525
                                526
                                527
                                528
                                529
                                530
                                531
                                532
                                533
                                534
                                535
                                536
                                537
                                538
                                539
                                540
                                541
                                542
                                543
                                544
                                545
                                546
                                547
                                548
                                549
                                550
                                551
                                552
                                553
                                554
                                555
                                556
                                557
                                558
                                559
                                560
                                561
                                562
                                563
                                564
                                565
                                566
                                567
                                568
                                569
                                570
                                571
                                572
                                573
                                574
                                575
                                576
                                577
                                578
                                579
                                580
                                581
                                582
                                583
                                584
                                585
                                586
                                587
                                588
                                589
                                590
                                591
                                592
                                593
                                594
                                595
                                596
                                597
                                598
                                599
                                600
                                601
                                602
                                603
                                604
                                605
                                606
                                607
                                608
                                609
                                610
                                611
                                612
                                613
                                614
                                615
                                616
                                617
                                618
                                619
                                620
                                621
                                622
                                623
                                624
                                625
                                626
                                627
                                628
                                629
                                630
                                631
                                632
                                633
                                634
                                635
                                636
                                637
                                638
                                639
                                640
                                641
                                642
                                643
                                644
                                645
                                646
                                647
                                648
                                649
                                650
                                651
                                652
                                653
                                654
                                655
                                656
                                657
                                658
                                659
                                660
                                661
                                662
                                663
                                664
                                665
                                666
                                667
                                668
                                669
                                670
                                671
                                672
                                673
                                674
                                675
                                676
                                677
                                678
                                679
                                680
                                681
                                682
                                683
                                684
                                685
                                686
                                687
                                688
                                689
                                690
                                691
                                692
                                693
                                694
                                695
                                696
                                697
                                698
                                699
                                700
                                701
                                702
                                703
                                704
                                705
                                706
                                707
                                708
                                709
                                710
                                711
                                712
                                713
                                714
                                715
                                716
                                717
                                718
                                719
                                720
                                721
                                722
                                723
                                724
                                725
                                726
                                727
                                728
                                729
                                730
                                731
                                732
                                733
                                734
                                735
                                736
                                737
                                738
                                739
                                740
                                741
                                742
                                743
                                744
                                745
                                746
                                747
                                748
                                749
                                750
                                751
                                752
                                753
                                754
                                755
                                756
                                757
                                758
                                759
                                760
                                761
                                762
                                763
                                764
                                765
                                766
                                767
                                768
                                769
                                770
                                771
                                772
                                773
                                774
                                775
                                776
                                777
                                778
                                779
                                780
                                781
                                782
                                783
                                784
                                785
                                786
                                787
                                788
                                789
                                790
                                791
                                792
                                793
                                794
                                795
                                796
                                797
                                798
                                799
                                800
                                801
                                802
                                803
                                804
                                805
                                806
                                807
                                808
                                809
                                810
                                811
                                812
                                813
                                814
                                815
                                816
                                817
                                818
                                819
                                820
                                821
                                822
                                823
                                824
                                825
                                826
                                827
                                828
                                829
                                830
                                831
                                832
                                833
                                834
                                835
                                836
                                837
                                838
                                839
                                840
                                841
                                842
                                843
                                844
                                845
                                846
                                847
                                848
                                849
                                850
                                851
                                852
                                853
                                854
                                855
                                856
                                857
                                858
                                859
                                860
                                861
                                862
                                863
                                864
                                865
                                866
                                867
                                868
                                869
                                870
                                871
                                872
                                873
                                874
                                875
                                876
                                877
                                878
                                879
                                880
                                881
                                882
                                883
                                884
                                885
                                886
                                887
                                888
                                889
                                890
                                891
                                892
                                893
                                894
                                895
                                896
                                897
                                898
                                899
                                900
                                901
                                902
                                903
                                904
                                905
                                906
                                907
                                908
                                909
                                910
                                911
                                912
                                913
                                914
                                915
                                916
                                917
                                918
                                919
                                920
                                921
                                922
                                923
                                924
                                925
                                926
                                927
                                928
                                929
                                930
                                931
                                932
                                933
                                934
                                935
                                936
                                937
                                938
                                939
                                940
                                941
                                942
                                943
                                944
                                945
                                946
                                947
                                948
                                949
                                950
                                951
                                952
                                953
                                954
                                955
                                956
                                957
                                958
                                959
                                960
                                961
                                962
                                963
                                964
                                965
                                966
                                967
                                968
                                969
                                970
                                971
                                972
                                973
                                974
                                975
                                976
                                977
                                978
                                979
                                980
                                981
                                982
                                983
                                984
                                985
                                986
                                987
                                988
                                989
                                990
                                991
                                992
                                993
                                994
                                995
                                996
                                997
                                998
                                999
                                1000

```

330188-83

270189-62

MCS-96 MACRO ASSEMBLER A2D_BUFFERING_UTILITY 02/18/86 PAGE 10

ERR	LOC	OBJECT	LINE	SOURCE STATEMENT
	00AC		332	CSEG
	00AC		333	
	00AC		334	A2D_DONE_Vector: ; A/D INTERRUPT ROUTINE
	00AC F2		335	PUSHF
	00AD C60600	E	337	STB ad_result_lo,[top_of_buffer]+
	00B0 C60600	E	338	STB ad_result_hi,[top_of_buffer]+
	00B3 51070900	E	339	ANDB ad_command,Control_A2D,\$00000111b ;load channel number
	00B7 E00809	R	341	DJNZ sample_count, Sample_Again
	00BA 1708	R	342	INCB sample_count
	00BC 880406	R	344	CMP top_of_buffer,aductempl ; Check top of buffer
	00BF DF26		345	BE Top_of_buffers
	00C1 F3		346	POPF
	00C2 F0		347	RET
	00C3		348	
	00C3 640A02	R	349	Sample_Again: ADD aductempl0,Sample_Period ; Set next sample time
	00C6 880406	R	351	CMP top_of_buffer,aductempl ; Check top of buffer
	00C7 13		352	JL Top_of_buffers ; for later jump
	00CF 300808	R	353	Load_HSO_Command_Start_A2D
	00D2	R	357	JBC sample_count,0,Make_HSO_High ; Jump here if carry or below
	00D2 FD		358	Make_HSO_low: JBC sample_count,0,Make_HSO_High ; Jump here if carry or below
	00D9 DFOC	E	361	nop ; wait 8 states after HSO load
	00DB F3		362	Load_HSO_Command_HSO_0_Low ; Load HSO Command HSO_0 Low
	00DC F0	E	366	BE Top_of_buffers ; Load for change of HSO to trigger S/H
	00DD		367	POPF
	00E3 DF02	E	368	RET ; End of routine
	00E5 F3		369	RET
	00E6 F0		370	RET
	00E7 717F09	R	371	Make_HSO_high: Load_HSO_Command_HSO_0_High ; Load for change of HSO to trigger S/H
	00EA F3	E	372	BE Top_of_buffers ; Load for change of HSO to trigger S/H
	00EB F0		376	POPF
	00EC	E	378	RET ; End of routine
	00E7		379	RET
	00E7 717F09	R	380	Top_of_buffers: ANDB Control_A2D,\$NOT(Busy) ; Clear converter BUSY bit
	00EA F3	E	382	POPF
	00EB F0		383	RET
	00EC	E	384	RET
	00EC	E	385	END

ASSEMBLY COMPLETED, NO ERROR(S) FOUND.

270189-63

The listing contains a fairly complete description of what the program does. The block by block operations are shown below:

Lines 1-198 describe the program, declare the variables and set up equates. Several of these variables are declared as overlayable, so the user needs to be careful if using this module for other than the FFT program.

Lines 205-210 declare a macro which is used to load the HSO unit. This will be used repeatedly through the code.

Lines 212-253 determine whether a conversion or download has been requested. If a download has been requested, the data is downloaded to the destination array as either paired or linear data. Paired data has been described earlier.

Lines 255-278 contain a subroutine which converts the destination array to either signed or unsigned numbers. The numbers are also shifted right to provide the desired full-scale value as requested by `SHIFT_COUNT`.

Lines 279-334 initialize the conversion routine. `HSO.0` is toggled with the start of each routine so that an external sample and hold can be used. The instructions in lines 308, 316, and 326 have been interweaved with the `Load_HSO_Commands` to provide the required 8 state delays between HSO loadings. If this was not done, NOPs would have been needed. It is easier to understand the code if these lines are thought of as being gathered at line 326.

Lines 337-353 are the actual A/D interrupt routine. The A/D results are placed BYTE by BYTE on the buffer, the A/D is reloaded, and then the number of samples taken is compared to the number needed. Note that the A/D command register needs to be reloaded even if the channel does not change. `INCB` on line 348 is used to insure that the `DJNZ` falls through on the next pass (if `sample_count` is not reset).

Lines 355-396 complete the routine. The HSO is set up to trigger the next conversion and provide the `HSO.0` toggle for an external sample and hold. Once again, the time between consecutive loads of the HSO is 8 states minimum. Note that this section of code has been optimized for speed by reducing branches to an absolute minimum and duplicating code where needed.

This concludes the description of the A to D buffer module. In the FFT program, this module is run, then the FFT transform module, then the plot module. This allows variables to be overlaid, saving RAM space. The time cost for this is not bad, considering the printer is the limiting factor in these conversions. If more RAM

was provided, and the FFT was run with its data in external RAM, this module could be run simultaneously with the other modules.

10.0 DATA PLOTTING MODULE

The plot module is relatively straight-forward, and is shown in Listing 5. After the declarations, which include overlayable registers, an initialization routine is listed. This separately called routine sets up the serial port on the 8096 to talk to the printer. In this case, the port has to be set for 300 baud.

A console out routine follows. This routine can also be called by any program, but it is used only by the plot routine in this example. The write to port 1 is used to trace the program flow. The character to be output is passed to this routine on the stack. This conforms to PLM-96 requirements.

Since all stack operations on the 8096 are 16-bits wide, a multiple character feature has been added to the console out routine. If the high byte it receives is non-zero, the ASCII character in that byte is printed after the character in the low byte. If the high byte has a value between 128 and 255, the character in the low byte is repeated the number of times indicated by the least significant 7 bits of the high byte.

The print decimal number routine is next. It is called with two words on the stack. The first word is the unsigned value to be printed. The second byte contains information on the number of places to be printed and zero and blank suppression. This routine is not overflow-proof. The user must declare a sufficient number of places to be printed for all possible numbers.

The `DRAW_GRAPH` routine provides the plot. It first sends a series of carriage return, line feeds (CRLFs) to clear the printer and provides a margin on the paper. Each row is started with the row number, 2 spaces, and a "+". Asterisks are then plotted until

Number of asterisks > FFT Value / `PLOT_RES`

Recall that `PLOT_RES` is a variable set by the main program. When the number of asterisks hits the desired value, the value of the line is printed. If the Decibel mode is selected, the line value is divided by 512 and printed in integer + decimal part form, followed by "dB". If the number of asterisks reaches `PLOT_MAX`, no value is printed. The next line is then started. A line with only a "!" is printed before the next plot line to provide a more aesthetic display on the printer. If a CRT was used, this extra line would probably not be wanted.

270189-64

PAGE 1

02/18/86

```

MCS-96 MACRO ASSEMBLER      PLOT_SERIAL
SERIES-III MCS-96 MACRO ASSEMBLER, V1.0

SOURCE FILE: :F2:PLOTSP.A96
OBJECT FILE: :F2:PLOTSP.OBJ
CONTROLS SPECIFIED IN INVOCATION COMMAND: NOSB

```

```

ERR LOC OBJECT          LINE      SOURCE STATEMENT
1      $pagelength(50)
2
3      PLOT_SERIAL MODULE STACKSIZE (6)
4
5      ; Intel Corporation, December 12, 1985
6      ; by Ira Horden, MCO Applications
7
8      ; This program produces a plot on serially connected printer. The
9      ; magnitude of each of the 32 input values is plotted horizontally, with one
10     ; "!" followed by a linefeed between each plot line. Each plot line starts
11     ; with a "+" and the entire plot begins with 3 line feeds and ends with a form
12     ; feed. The values to be plotted are 32 unsigned words based at the externally
13     ; defined pointer PLOT_IN.
14
15     ; The routine INIT_OUTPUT must be run to set up the serial port when the
16     ; system is turned on. CON_OUT can be used by a program to output to the
17     ; serial port. DRAW_GRAPH is the routine that automatically plots the data.
18
19     ; Sizing of the graph can be done using PLOT_RES, which determines how many
20     ; units are needed for each dot, and PLOT_MAX, which is the maximum value the
21     ; program will be passed. Note that (PLOT_MAX/PLOT_RES) defines the maximum
22     ; number of columns the routine will print.
23     ;
24
25     RSEG
26
27     EXTRN    iocl, baud_reg, spcon, spatat, sbuf, portl
28     EXTRN    zero, ax, bx, cx, dx, FFT_MODE
29     sptmp:   db      1
30
31     OSEG at 24H
32     value:   dsbl    1
33     divisor: dsbl    1
34     xptr:    dsw     1
35     yptr:    dsw     1
36     xval:    dsw     1
37     log_val: dsw     1
38
39     DSEG
40     EXTRN    PLOT_IN
41
42     $ject

```

Listing 5—The Plot Module

MCS-96 MACRO ASSEMBLER PLOT_SERIAL 02/18/86 PAGE 2

ERR	LOC	OBJECT	LINE	SOURCE STATEMENT
	2500		42	
			43	CSEG at 2500H ;;; PROGRAM MODULE BEGINS
			44	
			45	PUBLIC INIT_OUTPUT, CON_OUT, DRAW_GRAPH
			46	EXTRN PLOT_RES, PLOT_RES_2, PLOT_MAX
	2500		47	
			48	INIT_OUTPUT: ; INITIALIZE SERIAL PORT
			49	
	2500 B12000	E	50	ldi iocl,#00100000B ; set p2.0 to txd
	0270		51	
	0082		52	baud_val equ 624 ; 624=300 baud (at 12 MHz)
	006F		53	
			54	baud_high equ ((baud_val-1)/256) OR 80H ; set for XTAL1 clock
			55	baud_low equ (baud_val-1) MOD 256
	2503 B16F00	E	56	
	2506 B18200	E	57	ldi baud_reg,#baud_low
	2509 B14900	E	58	ldi baud_reg,#baud_high
	250C B12000	R	59	
			60	spcon,#01001001b ; enable receiver mode 1
			61	sptmp,#00100000B ; set TI-tmp
	250F F0		62	
			63	RET
			64	
			65	\$object
			66	
			67	
			68	
			69	
			70	
			71	
			72	
			73	
			74	
			75	
			76	
			77	
			78	
			79	
			80	
			81	
			82	
			83	
			84	
			85	
			86	
			87	
			88	
			89	
			90	
			91	
			92	
			93	
			94	
			95	
			96	
			97	
			98	
			99	
			100	

END OF MACRO ASSEMBLER PLOT_SERIAL 02/18/86 PAGE 2

MCS-96 MACRO ASSEMBLER PLOT_SERIAL

02/18/86

PAGE 3

```

ERR LOC OBJECT      LINE      SOURCE STATEMENT
66
67 ;
68 ;
69 ;
70 ;
71 ;
72 ;
73 ;
74 ;
75 ;
76 ;
77 ;
78 CON_OUT:
79     pop     ax             ; cx contains the calling adress
80     pop     dx
81     jbs     dx+1,7,onechr   ; If bit 7 is set print one character
82     cmpb    dx+1,zero      ;
83     je      onechr         ; if highbyte=0 print one character
84
85     twochr: orb     sptmp,sptat ; wait for TI
86             jbc     sptmp,5,twochr
87             andb    sptmp,#11011111b ; clear TI-tmp
88             orb     zero,sptat ; remove possible false TI
89
90     2528 B00000     E 90     ldb     sbuf,dx
91     252B B00100     E 91     ldb     dx,dx+1 ; Load second character
92     252E 1101     E 92     clrb    dx+1 ; clear count byte
93     2530 717F00     E 93     andb    dx,#07FH ; mask MSB
94
95     2533 1701     E 95     onechr: incb  dx+1 ;
96     2535 717F01     E 96     andb    dx+1,#7FH
97     2538 900000     E 97     wait1: orb     sptmp,sptat ; wait for TI
98     253B 3500FA     R 98     jbc     sptmp,5,wait1
99     253E 71DF00     R 99     andb    sptmp,#11011111b ; clear TI-tmp
100    2541 900000     E 100    orb     zero,sptat ; remove possible false TI
101
102    2544 B00000     E 102    ldb     sbuf,dx
103    2547 B001FE     E 103    jnz     dx+1,wait1 ;
104    254A E300     E 104     BR      [ax] ; Effectively a RET
105
106    2550 B13000     E 106    $eject

```

270189-66


```

ERR LOC OBJECT          LINE      SOURCE STATEMENT
                                107 ;
                                108 ;
                                109 ;
                                110 ; Call with two words on stack. The first is the value to be printed.
                                111 ; The second has mode information in the low byte.
                                112 ; MODE: 000 = suppress all zeros
                                113 ; 001 = print all numbers
                                114 ; 010 = suppress all zeros except rightmost
                                115 ; 1xx = do not print leading blanks
                                116 ;
                                117 ; The high byte of the 2nd word = 2x the number of places to be printed
                                118 ;
                                119 ;
254C 254C 120 PRINT_NUM: ; Send Decimal number to CON_OUT
254C CC00 121 pop cx ;
254E CC00 122 pop bx ; bx is mode byte, bx+1 is divisor pointer
2550 AC0100 123 ldhcx dx,bx+1
2553 A300962528 124 ld divisor,divtab[dx]
2558 CC24 125 pop value
255A 126 div_loop:
255A 0126 127 clr value+2
255C 8C2824 128 divu value,divisor ; divide ax,dx by divisor
255F 380017 129 jbs bx,0,chr_ok ; print character regardless of value
2562 980024 130 cmpr value,zero ;
2565 D70F 131 jne non_0 ; jump if value is non zero
2567 132 Val_0: ; Value is zero
2567 310003 133 jbc bx,1,prntsp ; Print space instead of 0
256A 38280C 134 jbs divisor,0,chr_ok ; If in rightmost position print 0
256D 3A0015 135 prntsp: jbs bx,2,cont ; Do not print space if bit is set
2570 A1F00024 136 ld value,#0F0H ; 0F0h+30h = 20H = space
2574 2003 137 br chr_ok
2576 910100 138 non_0:
2576 910100 139 non_0: orb bx,#0001B ; Set flag so 0's will be printed
2579 65300024 140 chr_ok: add value,#30h ; 30h + n = 0 to 9 ascii
257D 617F0024 141 and value,#7Fh ; send least sig seven bits, clear upper word
2581 C824 142 push value
2583 2F8B 143 call con_out ; output ascii result (result<9)
2585 A02624 144 cont: ld value,value+2 ; load value with remainder
2588 012A 145 clr divisor+2
258A 8D0A0028 146 divu divisor,#10 ; next lower power of ten
258E 890028 147 cmp divisor,zero
2591 D7C7 148 jne div_loop
2593 149 div_done:
2593 E300 150 br [cx]
2596 151
2596 152 DIVTAB: ; Number of places for result
2596 000001000A006400 153 dcw 0, 1, 10, 100, 1000, 10000 ; divisor table - 10**n

```

510189-28

270189-67

PAGE 5

270189-68

Listing 5—The Plot Module (Continued)

6-158

MCS-96 MACRO ASSEMBLER PLOT_SERIAL

02/18/86

PAGE 6

```

ERR LOC OBJECT          LINE    SOURCE STATEMENT
                                198
                                199
                                PRT_NUM:
25EC          200          cmp     yptr,#PLOT_RES_2      ; If value is less then minimum needed
25EC 8900002E          201          be      NXTLN          ; for a plot, do not print value
25F0 DF49          202
                                203          push    #2020H          ; print 2 spaces then value
25F2 C92020          204          call    con_out
25F5 2F19          205          JBS     FFT_MODE,3,db_mode
25F7 38000B          206
                                207          norm_mode:
25FA          208          push    PLOT_IN[xptr]
25FA CB2D0000          209          push    #(0A00H or 0000B)      ; supress all zeros
25FE C9000A          210          call    PRINT_NUM
2601 2F49          211          BR      NXTLN
2603 2036          212
                                213          db_mode:
2605          214          ld      yptr,plot_in[xptr]      ; PLOT_IN = 512*10*LOG(x)
2605 A32D00002E          215          shr     yptr,#1          ; yptr=265 * 10LOG(x)
260A 08012E          216          ldhze   ax,yptr+1          ; ax= 10LOG(x) = yptr/256
260D AC2F00          217
                                218          push    ax          ; Print AX
2610 C800          219          push    #(0A00H or 0010B)      ; supress all but rightmost zero
2612 C9020A          220          call    PRINT_NUM
2615 2F35          221          push    #2EH          ; Decimal point
2617 C92E00          222          call    con_out
261A 2EF4          223
                                224          ld      ax+1,yptr      ; high byte of ax = fractional portion of
261C B02E01          225          clrb     ax          ; 10LOG(x)
261F 1100          226
                                227          mulu    ax,#3E6H      ; if ax=FF00H then ax+2 now = 998 decimal
2621 6DE60300          228          jbc     ax+1,7,no_rnd
2625 370102          229          inc     dx          ; round value up
2628 0700          230
                                231          no_rnd: push    dx          ; dx= ax+2
262A C800          232          push    #(600H or 0001B)      ; print all numbers to three places
262C C90106          233          call    Print_num
262F 2F1B          234          push    #20H          ; space
2631 C92000          235          call    con_out
2634 2EDA          236          push    #4264H          ; "dB"
2636 C96442          237          call    con_out
2639 2ED5          238
                                239          $eject

```

S10189-10

270189-69

ERR LOC	OBJECT	LINE	SOURCE STATEMENT
		240	
		241	
		242	NXTLN: push #0A0DH ; Setup for next line
		243	call CON_OUT ; CRLF
		244	push #00H ; nul
		245	call CON_OUT
		246	push #8620H ; 7 spaces
		247	call CON_OUT
		248	push #21H ; !
		249	call con_out ; !
		250	
		251	inc xval ;
		252	add xptr,#2
		253	cmp xptr,#62
		254	ble nrt_row ; Start printing next row
		255	
		256	Done: push #0A0DH ; CRLF ; Form feed for next graph
		257	call CON_OUT
		258	push #0C00H ; null,FF
		259	call con_out
		260	
		261	RET
		262	END
		263	
		264	
		265	
		266	
		267	
		268	
		269	
		270	
		271	
		272	
		273	
		274	
		275	
		276	
		277	
		278	
		279	
		280	
		281	
		282	
		283	
		284	
		285	
		286	
		287	
		288	
		289	
		290	
		291	
		292	
		293	
		294	
		295	
		296	
		297	
		298	
		299	
		300	
		301	
		302	
		303	
		304	
		305	
		306	
		307	
		308	
		309	
		310	
		311	
		312	
		313	
		314	
		315	
		316	
		317	
		318	
		319	
		320	
		321	
		322	
		323	
		324	
		325	
		326	
		327	
		328	
		329	
		330	
		331	
		332	
		333	
		334	
		335	
		336	
		337	
		338	
		339	
		340	
		341	
		342	
		343	
		344	
		345	
		346	
		347	
		348	
		349	
		350	
		351	
		352	
		353	
		354	
		355	
		356	
		357	
		358	
		359	
		360	
		361	
		362	
		363	
		364	
		365	
		366	
		367	
		368	
		369	
		370	
		371	
		372	
		373	
		374	
		375	
		376	
		377	
		378	
		379	
		380	
		381	
		382	
		383	
		384	
		385	
		386	
		387	
		388	
		389	
		390	
		391	
		392	
		393	
		394	
		395	
		396	
		397	
		398	
		399	
		400	
		401	
		402	
		403	
		404	
		405	
		406	
		407	
		408	
		409	
		410	
		411	
		412	
		413	
		414	
		415	
		416	
		417	
		418	
		419	
		420	
		421	
		422	
		423	
		424	
		425	
		426	
		427	
		428	
		429	
		430	
		431	
		432	
		433	
		434	
		435	
		436	
		437	
		438	
		439	
		440	
		441	
		442	
		443	
		444	
		445	
		446	
		447	
		448	
		449	
		450	
		451	
		452	
		453	
		454	
		455	
		456	
		457	
		458	
		459	
		460	
		461	
		462	
		463	
		464	
		465	
		466	
		467	
		468	
		469	
		470	
		471	
		472	
		473	
		474	
		475	
		476	
		477	
		478	
		479	
		480	
		481	
		482	
		483	
		484	
		485	
		486	
		487	
		488	
		489	
		490	
		491	
		492	
		493	
		494	
		495	
		496	
		497	
		498	
		499	
		500	

270189-70

At the end of the plot, a form feed is given to set the printer up for the next graph. Our printer would frequently miss the character after a CRLF. To solve this problem, a null (ASCII 0) is sent after every CRLF to make sure the printer is ready for the next line. This has been found to be a problem with many devices running at close to their maximum capacity, and the nulls work well to solve it.

With the plot completed, the program begins to run again by taking another set of A to D samples.

11.0 USING THE FFT PROGRAM

The program can be used with either real or tabled data. If real data is used, the signal is applied to analog channel 1. The program as written performs A/D samples at 100 microsecond intervals, collecting the 64 samples in 6.4 milliseconds. This sets the sampling window frequency at 156 Hz. If tabled data is used, 64 words of data should be placed in the location pointed to by DATA0 in the TABLE_LOAD routine of the Main Module.

Program control is specified by FFT_MODE which is loaded in the main module. Also within the main module are settings which control the A to D buffer routine and the Plot routine. The intention was to have only one module to change and recompile to vary parameters in the entire program.

The program modules are set up to run one-at-a-time so that the code would be easy to understand. Additionally, the Plot routine takes so long relative to the other sections, that it doesn't pay to try to overlap code sections. If this code were to be converted to run a process instead of print a graph, it might be worthwhile to run the FFT and the A/D routines at the same time.

If the goal of a modified program is to have the highest frequency sampling possible, it might be desirable to streamline the A/D section and run it without interruption. When the A to D routine was complete the FFT routine could be started. The reasoning behind this is that at the fastest A/D speeds the processor will be almost completely tied up processing the A/D information and storing it away. Using an interrupt based A/D routine would slow things down.

A set of programs which will perform a FFT has been presented in this application note. These programs are available from the INSITE users library as program CA-26. More importantly, dozens of programing examples have been made available, making it easier to get started with the 8096. Examples of how to use the hardware on the 8096 have already appeared in AP-248, "Using The 8096". These two applications notes form a good base for the understanding of MCS-96 microcontroller based design.

$$N \cdot \text{EXPONENT} \quad \text{complex multiplications}$$

$$\text{and}$$

$$N \cdot \text{EXPONENT} \quad \text{complex additions}$$

where

$$\text{EXPONENT} = \log_2 N$$

A standard Fourier Transform requires

$$N^2 \text{ complex multiplications}$$

and

$$N(N-1) \text{ complex additions}$$

The matrix can be reduced to have less non-trivial multiplications

$$\begin{bmatrix} Y(0) \\ Y(1) \\ Y(2) \\ Y(3) \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & W_4 & W_4^3 & W_4^2 \\ 1 & W_4^2 & W_4^4 & W_4^6 \\ 1 & W_4^3 & W_4^5 & W_4^7 \end{bmatrix} \begin{bmatrix} X(0) \\ X(1) \\ X(2) \\ X(3) \end{bmatrix}$$

The square matrix can be factored into

$$\begin{bmatrix} Y(0) \\ Y(2) \\ Y(1) \\ Y(3) \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & W_4 & W_4^2 & W_4^3 \\ 1 & W_4^2 & W_4^4 & W_4^6 \\ 1 & W_4^3 & W_4^5 & W_4^7 \\ 1 & W_4^4 & W_4^6 & W_4^7 \end{bmatrix} \begin{bmatrix} X(0) \\ X(1) \\ X(2) \\ X(3) \end{bmatrix}$$

For this equation to work the Y(1) and Y(2) terms need to be swapped, as shown above. This procedure is a bit unusual, as described in the text.

12.0 APPENDIX A - MATRICES

Matrices are a convenient way to express groups of equations. Consider the complex discrete Fourier Transform in equation 9, with $N = 4$.

$$Y_n = \sum_{k=0}^3 X(k) W^{nk} \quad n = 0, 1, 2, 3$$

This can be expanded to

$$\begin{aligned} Y(0) &= X(0) W^0 + X(1) W^0 + X(2) W^0 + X(3) W^0 \\ Y(1) &= X(0) W^0 + X(1) W^1 + X(2) W^2 + X(3) W^3 \\ Y(2) &= X(0) W^0 + X(1) W^2 + X(2) W^4 + X(3) W^6 \\ Y(3) &= X(0) W^0 + X(1) W^3 + X(2) W^6 + X(3) W^9 \end{aligned}$$

In matrix notation, this is shown as

$$\begin{bmatrix} Y(0) \\ Y(1) \\ Y(2) \\ Y(3) \end{bmatrix} = \begin{bmatrix} W^0 & W^0 & W^0 & W^0 \\ W^0 & W^1 & W^2 & W^3 \\ W^0 & W^2 & W^4 & W^6 \\ W^0 & W^3 & W^6 & W^9 \end{bmatrix} \begin{bmatrix} X(0) \\ X(1) \\ X(2) \\ X(3) \end{bmatrix}$$

The first step to simplifying this is to reduce the center matrix. Recalling that

$$W^N = W^{N \bmod N} \quad \text{and} \quad W^0 = 1$$

The matrix can be reduced to have less non-trivial multiplications.

$$\begin{bmatrix} Y(0) \\ Y(1) \\ Y(2) \\ Y(3) \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & W^1 & W^2 & W^3 \\ 1 & W^2 & W^0 & W^2 \\ 1 & W^3 & W^2 & W^1 \end{bmatrix} \begin{bmatrix} X(0) \\ X(1) \\ X(2) \\ X(3) \end{bmatrix}$$

The square matrix can be factored into

$$\begin{bmatrix} Y(0) \\ Y(2) \\ Y(1) \\ Y(3) \end{bmatrix} = \begin{bmatrix} 1 & W^0 & 0 & 0 \\ 1 & W^2 & 0 & 0 \\ 0 & 0 & 1 & W^1 \\ 0 & 0 & 1 & W^3 \end{bmatrix} \begin{bmatrix} 1 & 0 & W^0 & 0 \\ 0 & 1 & 0 & W^0 \\ 1 & 0 & W^2 & 0 \\ 0 & 1 & 0 & W^2 \end{bmatrix} \begin{bmatrix} X(0) \\ X(1) \\ X(2) \\ X(3) \end{bmatrix}$$

For this equation to work, the $Y(1)$ and $Y(2)$ terms need to be swapped, as shown above. This procedure is a Bit Reversal, as described in the text.

Multiplying the two rightmost matrices results in

$$\begin{aligned} X(0) + X(2) W^0 \\ X(1) + X(3) W^0 \end{aligned} \quad \begin{aligned} &\text{requiring 4 complex multiplications} \\ &\text{X(0) + X(2) W}^3 \quad \text{\& 4 complex additions} \\ &\text{X(1) + X(3) W}^2 \end{aligned}$$

Noting that $W^0 = -W^2$, 2 of the complex multiplications can be eliminated, with the following results

$$\begin{aligned} X(0) + X(2) W^0 \\ X(1) + X(3) W^0 \end{aligned} \quad \begin{aligned} &\text{requiring 2 complex multiplications} \\ &\text{X(0) - X(2) W}^0 \quad \text{and 4 complex additions} \\ &\text{X(1) - X(3) W}^0 \end{aligned}$$

Since $W^1 = -W^3$, a similar result occurs when this vector is multiplied by the remaining square matrix.

The resulting equations are:

$$\begin{aligned} Y(0) &= (X(0) + X(2) W^0) + W^0 (X(1) + X(3) W^0) \\ Y(2) &= (X(0) + X(2) W^0) - W^0 (X(1) + X(3) W^0) \\ Y(1) &= (X(0) - X(2) W^0) + W^1 (X(1) - X(3) W^0) \\ Y(3) &= (X(0) - X(2) W^0) - W^1 (X(1) - X(3) W^0) \end{aligned}$$

The number of complex multiplications required is 4, as compared with 16 for the unfactored matrix.

In general, the FFT requires

$$\frac{N * \text{EXPONENT}}{2} \text{ complex multiplications}$$

and

$$N * \text{EXPONENT} \text{ complex additions}$$

where

$$\text{EXPONENT} = \log_2 N$$

A standard Fourier Transform requires

$$N^2 \text{ complex multiplications}$$

and

$$N(N-1) \text{ complex additions}$$

13.0 APPENDIX B - PLOTS

The following plots are examples of output from the FFT program. These plots were generated using tabled data, but very similar plots have also been made using the analog input module. Typically, a plot made using the analog input module will not show quite as much power at each frequency and will show a positive value for the DC component. This is because it is difficult to get exactly a full-scale analog input with no DC offset.

Plot 1 is a Magnitude plot of a square wave of period NT.

Plot 2 is the same data plotted in dB. Note how the dB plot enhances the difference in the small signal values at the high frequencies.

Plot 3 shows the windowed version of this data. Note that the widening of the bins due to windowing shows energy in the even harmonics that is not actually present. For data of this type a different window other than Hanning would normally be used. Many window types are available, the selection of which can be determined by the type of data to be plotted.³

Plot 4 shows a sine wave of period NT/7 or frequency 7/NT.

Plot 5 shows the same input with windowing. Note the signal shown in bins 6 and 8.

Plot 6 shows a sine wave of period NT/7.5. Note the noise caused by the discontinuity as discussed earlier.

Plot 7 uses windowing on the data used for plot 6. Note the cleaner appearance.

Plot 8 shows a sine wave input of magnitude 0.707 and period NT/7.5.

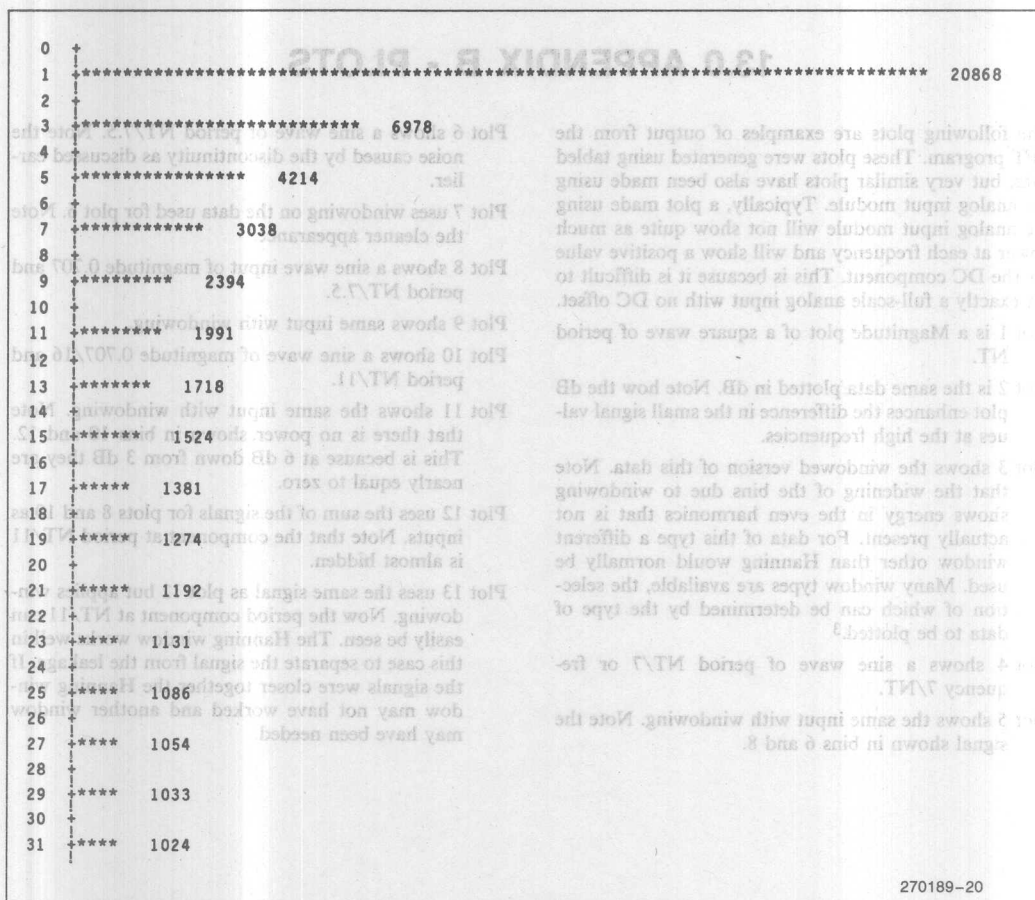
Plot 9 shows same input with windowing.

Plot 10 shows a sine wave of magnitude 0.707/16 and period NT/11.

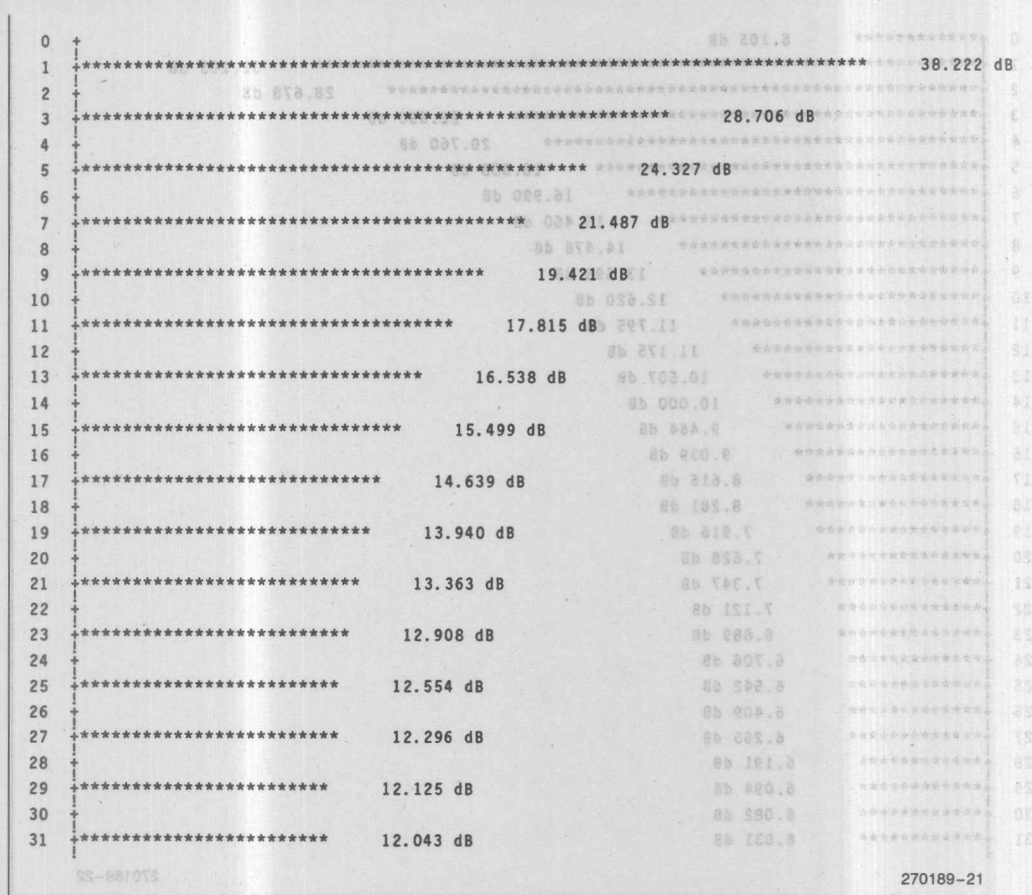
Plot 11 shows the same input with windowing. Note that there is no power shown in bins 10 and 12. This is because at 6 dB down from 3 dB they are nearly equal to zero.

Plot 12 uses the sum of the signals for plots 8 and 10 as inputs. Note that the component at period NT/11 is almost hidden.

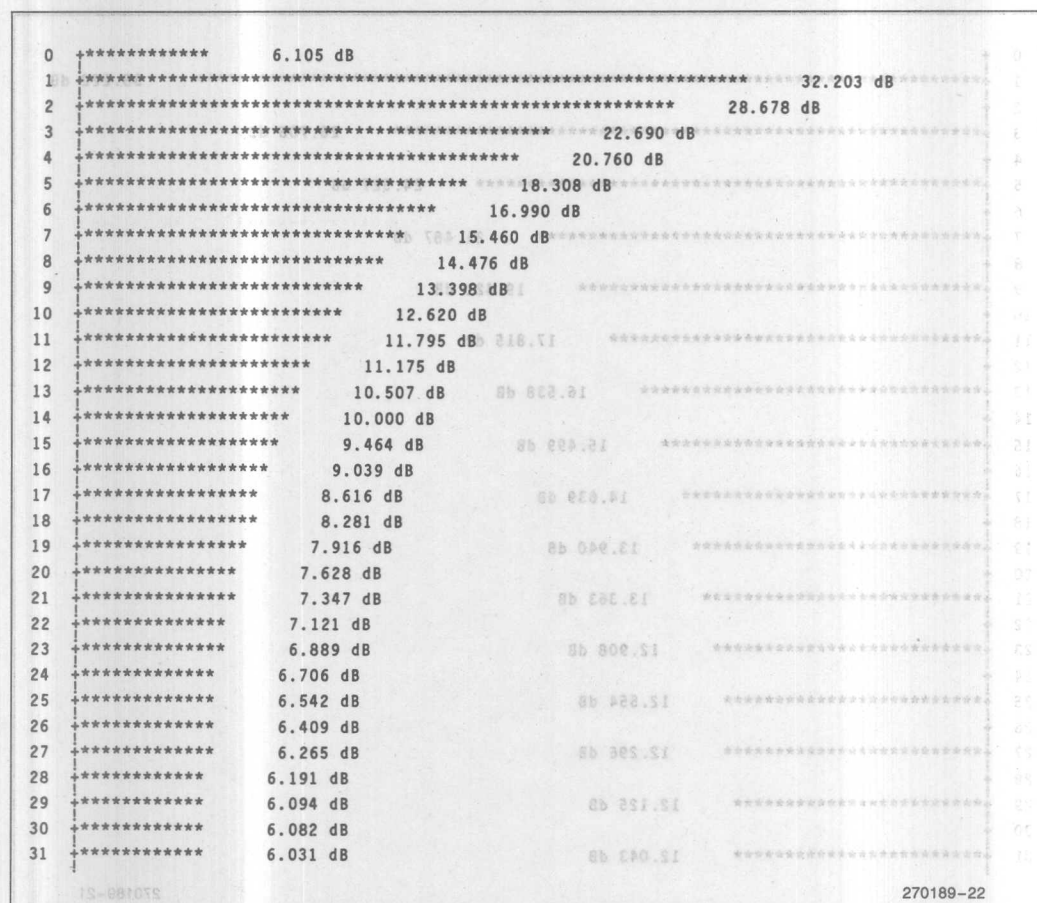
Plot 13 uses the same signal as plot 12 but applies windowing. Now the period component at NT/11 can easily be seen. The Hanning window works well in this case to separate the signal from the leakage. If the signals were closer together the Hanning window may not have worked and another window may have been needed.



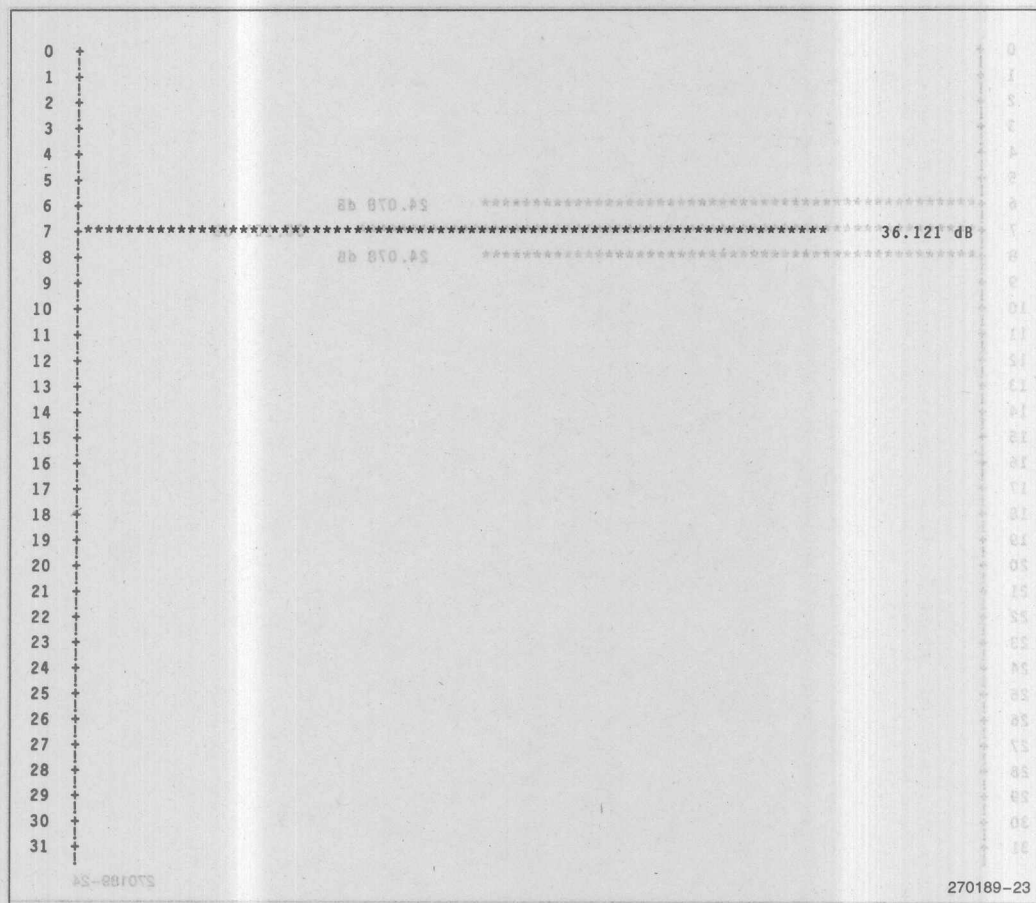
Plot 1—Magnitude Plot of Squarewave



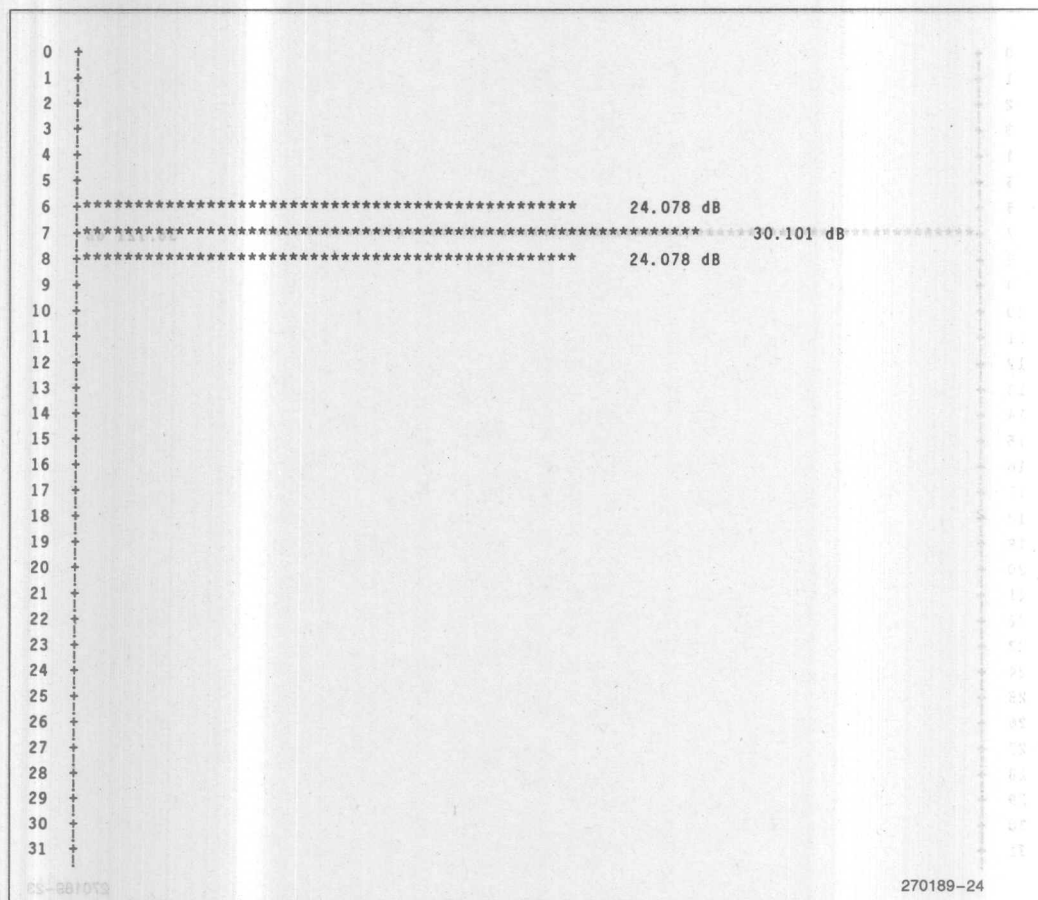
Plot 2—Decibel Plot of Squarewave



Plot 3—Plot of Squarewave with Window



Plot 4—Sin (7.0X) without Window



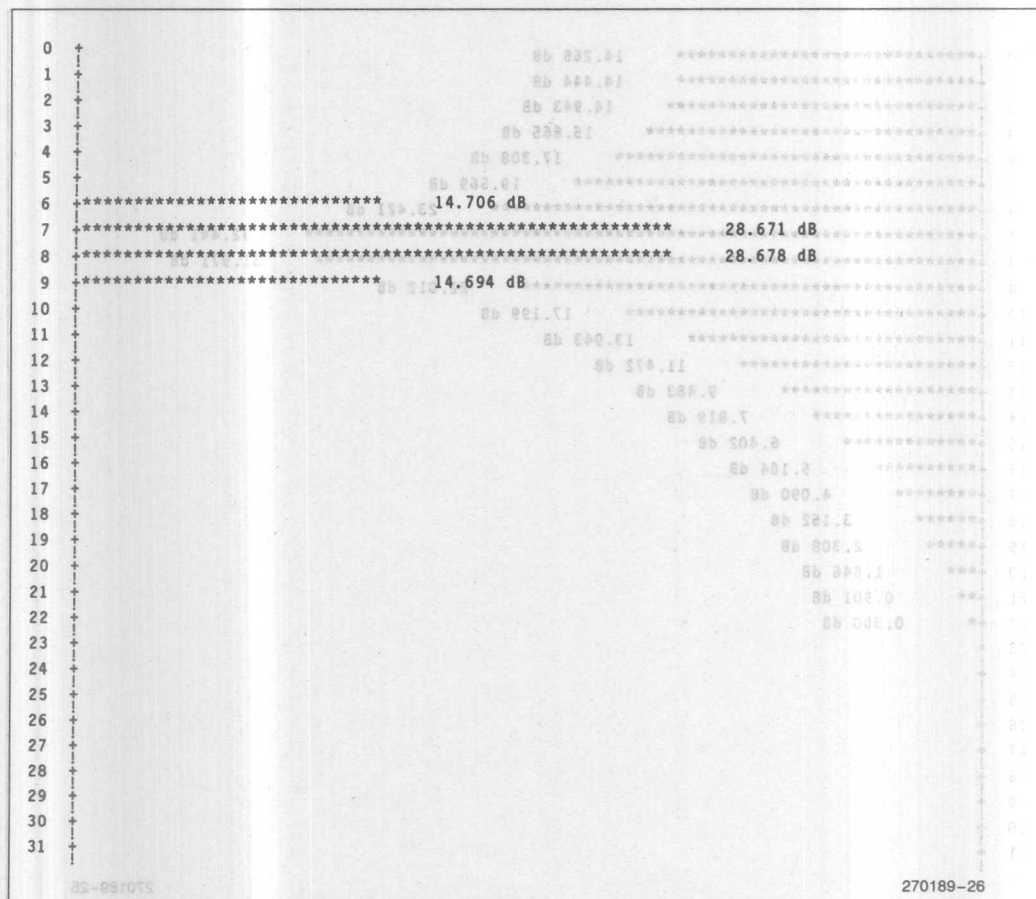
Plot 5—Sin (7.0X) with Window

0	*****	14.265 dB	0
1	*****	14.444 dB	1
2	*****	14.943 dB	2
3	*****	15.865 dB	3
4	*****	17.308 dB	4
5	*****	19.569 dB	5
6	*****	23.421 dB	6
7	*****	32.441 dB	7
8	*****	31.971 dB	8
9	*****	22.012 dB	9
10	*****	17.199 dB	10
11	*****	13.943 dB	11
12	*****	11.472 dB	12
13	*****	9.483 dB	13
14	*****	7.819 dB	14
15	*****	6.402 dB	15
16	*****	5.164 dB	16
17	*****	4.090 dB	17
18	*****	3.152 dB	18
19	*****	2.308 dB	19
20	****	1.546 dB	20
21	***	0.901 dB	21
22	**	0.300 dB	22
23	+		23
24	+		24
25	+		25
26	+		26
27	+		27
28	+		28
29	+		29
30	+		30
31	+		31

95-0810'S

270189-25

Plot 6—Sin (7.5X) without Window

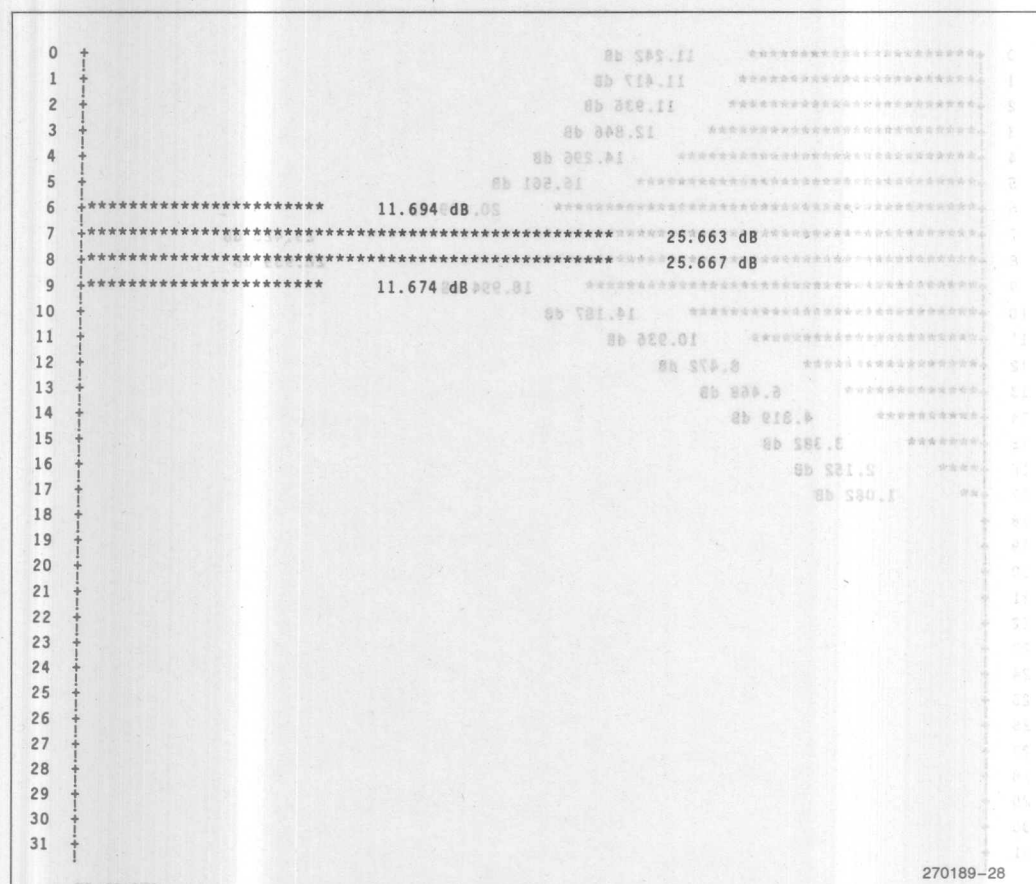


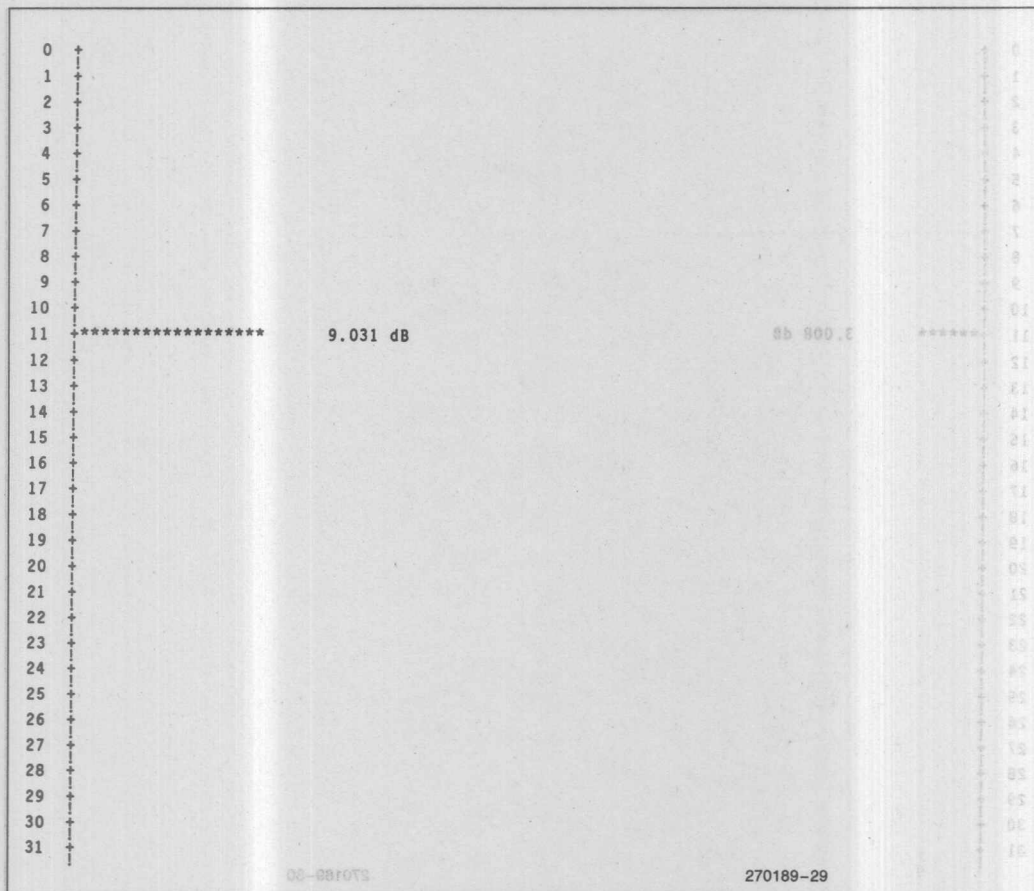
0	*****	11.242 dB	
1	*****	11.417 dB	
2	*****	11.936 dB	
3	*****	12.846 dB	
4	*****	14.296 dB	
5	*****	16.561 dB	
6	*****	20.409 dB	*****
7	*****		29.425 dB *****
8	*****		28.959 dB *****
9	*****	18.994 dB	*****
10	*****	14.187 dB	
11	*****	10.936 dB	
12	*****	8.472 dB	
13	*****	6.468 dB	
14	*****	4.819 dB	
15	*****	3.382 dB	
16	****	2.152 dB	
17	**	1.082 dB	
18	+		
19	+		
20	+		
21	+		
22	+		
23	+		
24	+		
25	+		
26	+		
27	+		
28	+		
29	+		
30	+		
31	+		

85-681055

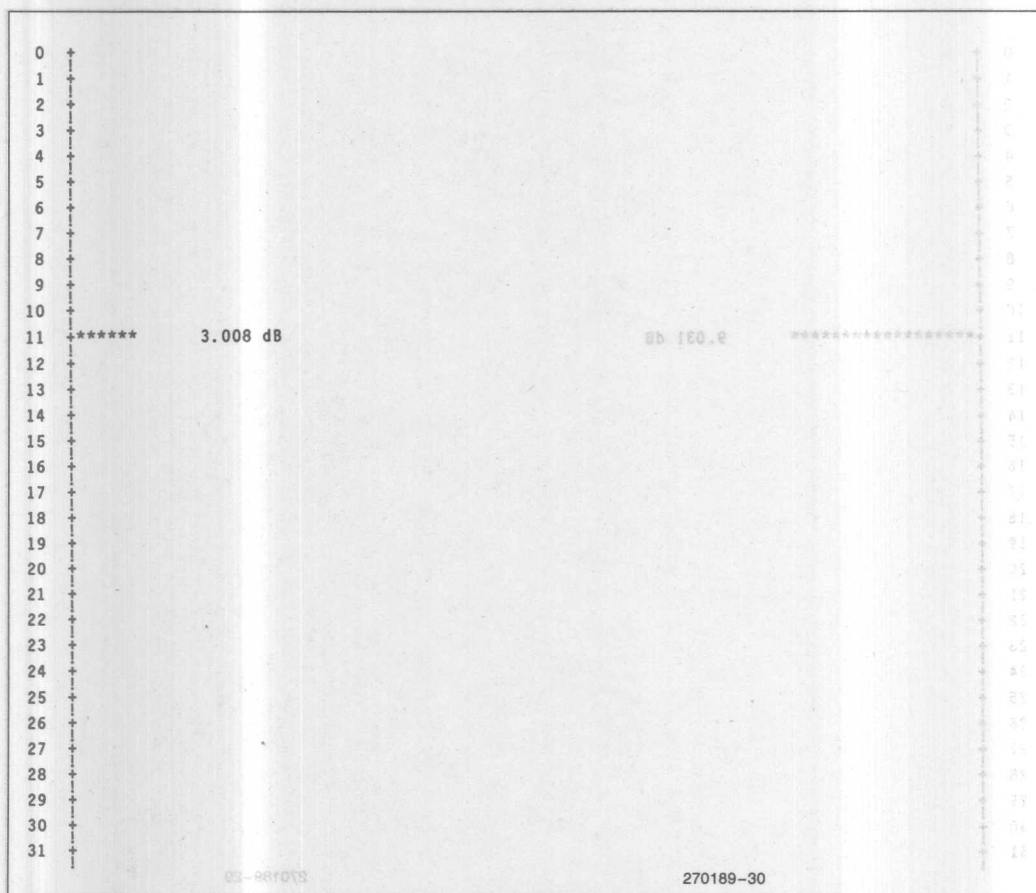
270189-27

Plot 8—0.707 * Sin (7.5X) without Window





Plot 10— $0.707/16 * \sin(11X)$ without Window



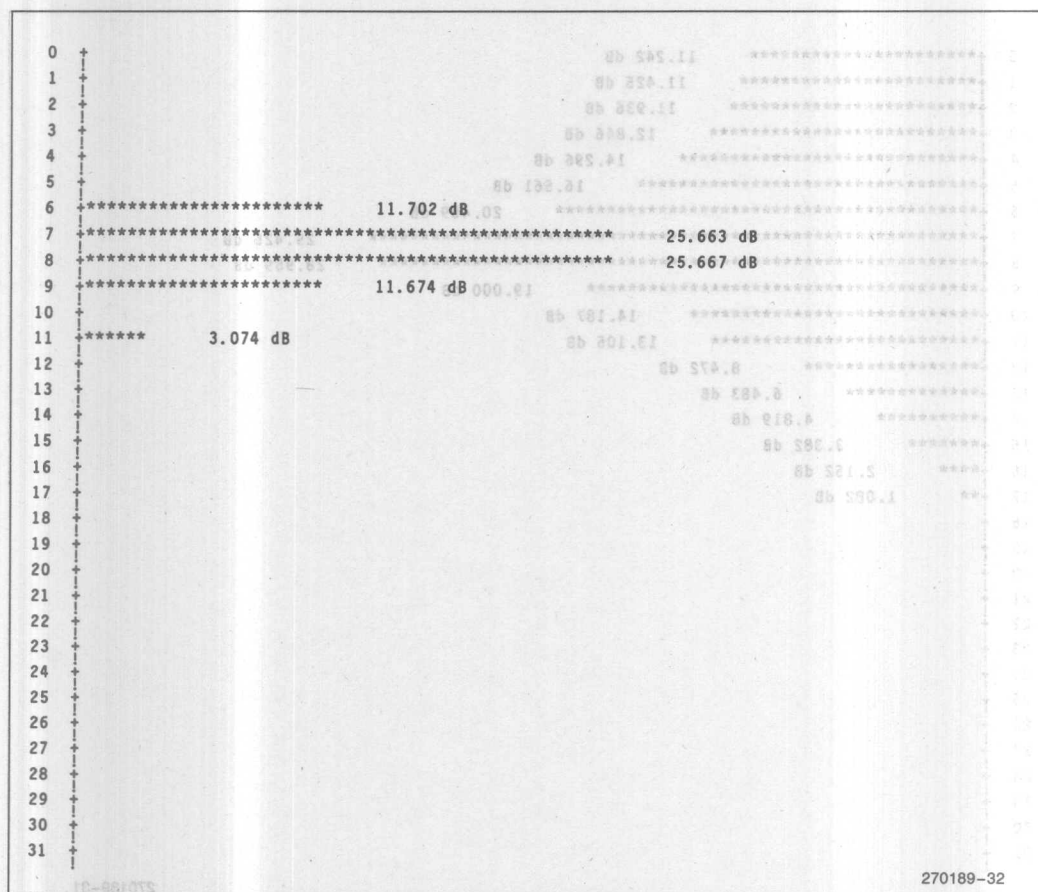
Plot 11-- $0.707/16 * \sin(11X)$ with Window

0	*****	11.242 dB	
1	*****	11.425 dB	
2	*****	11.936 dB	
3	*****	12.846 dB	
4	*****	14.296 dB	
5	*****	16.561 dB	
6	*****	20.409 dB	
7	*****	29.425 dB	
8	*****	28.959 dB	
9	*****	19.000 dB	
10	*****	14.187 dB	
11	*****	13.105 dB	
12	*****	8.472 dB	
13	*****	6.483 dB	
14	*****	4.819 dB	
15	*****	3.382 dB	
16	*****	2.152 dB	
17	**	1.082 dB	
18	+		
19	+		
20	+		
21	+		
22	+		
23	+		
24	+		
25	+		
26	+		
27	+		
28	+		
29	+		
30	+		
31	+		

31-081075

270189-31

Plot 12—0.707 (Sin (7.5X) + 1/16 Sin (11X)) without Window



270189-32

BIBLIOGRAPHY

1. Boyet, Howard and Katz, Ron, The 16-Bit 8096: Programming, Interfacing, Applications. 1985, Microprocessor Training Inc., New York, NY.
2. Brigham, E. Oran, The Fast Fourier Transform. 1974, Prentice-Hall, Inc., Englewood Cliffs, New Jersey.
3. Harris, Fredric J., On the use of Windows for Harmonic Analysis with the Discrete Fourier Transform. Proceedings of the IEEE, Vol. 66, No. 1, January 1978.
4. Weaver, H. Joseph, Applications of discrete and continuous Fourier analysis. 1983, John Wiley and Sons, New York.

INTEL PUBLICATIONS

1. 1986 Microcontroller Handbook, Order Number 210918-004
2. Using the 8096, AP-248, Order Number 270061-001
3. MCS-96 Macro Assembler User's Guide, Order Number 122048-001
4. MCS-96 Utilities User's Guide, Order Number 122049-001



APPLICATION BRIEF

AB-32

INTEL PUBLICATIONS
1. 1986 Microcontroller Handbook, Order Number 210918-004
2. Using the 8096, AB-348, Order Number 270521-001
3. MCS-96 Macro Assembler User's Guide, Order Number 133048-001
4. MCS-96 Utilities User's Guide, Order Number 133049-001

BIBLIOGRAPHY
1. Poyet, Howard and Katz, Ron, The 16-Bit 8096 Programming, Intertec, Applications, 1985, Microprocessor Training Inc., New York, NY.
2. Brigham, E. Oren, The Fast Fourier Transform, 1974, Prentice-Hall, Inc., Englewood Cliffs, New Jersey.
3. Hartley, Fredric J., On the use of Windows for Harmonic Analysis with the Discrete Fourier Transform, Proceedings of the IEEE, Vol. 66, No. 1, January 1978.
4. Weaver, H. Joseph, Applications of discrete and continuous Fourier analysis, 1981, John Wiley and Sons, New York.

December 1987

Upgrade Path from 8096-90 to 8096BH to 80C196

Converting applications that use an 8X9X-90 to use an 8X9XBH requires consideration of a few of the BH enhancements. Descriptions of each of the differences between the -90 and the BH follow, along with a discussion of the implications of the change.

BHE and INST are latched: The bus control signals **BHE** and **INST** are valid throughout the bus cycle on 8X9XBH devices. On -90 devices, these signals need to be latched on the falling edge of **ALE**.

Byte Read following RESET rising: The bus control and buswidth options of 8X9XBH devices are selected by configuration of the chip immediately following the rising edge of **RESET**. During the usual 10 state reset sequence, BH parts will perform a byte read of location 2018H to acquire configuration information prior to fetching the first opcode at location 2080H. The 8X9X-90 does not perform this read.

ALE is high while in reset: The **ALE/ADV** pin of the 8X9XBH is driven high while the **RESET** pin is held low. On -90 devices, **ALE** is driven low while in **RESET**. Circuits which rely on the state of **ALE** while **RESET** is low must be modified. The reset state of **ALE** was changed to enable implementation of the Chip Configuration Byte read from external memory following the rising edge of **RESET**.

EA is latched on RESET rising: The 8X9XBH latches the value of **EA** on the rising edge of **RESET**. On -90 devices, **EA** was not latched and could be changed without placing the part in **RESET**. This change was necessary to enhance ROM/EPROM security. Circuits that rely on **EA** not being latched must be modified.

A/D speed increased: The 8X95BH and 8X97BH A/D converters complete conversion in 88 state times. On -90 devices with A/D converters, a conversion takes 168 state times. This translates in an increased conversion speed from 42 μ s on -90 parts to 22 μ s on BH parts running at 12MHz. Software that relies upon the speed of conversion for timing must be changed. It is also recommended that MCS-96 software be written so as to not be impacted by further changes in A/D conversion speed.

Sample/Hold on A/D: The 8X95BH and 8X97BH have a sample/hold on the input of the A/D converter. 8X9X-90 devices with A/D converters do not have sample/hold circuitry. External analog circuitry which also includes a sample/hold must provide a settled analog input within the first four state times of 8X9XBH conversion.

Duplicate Fetches: The 8X9XBH bus controller was made more aggressive when it comes to instruction fetches in order to minimize the execution speed degra-

dation of using an 8-bit bus. As a result, instruction fetches over a 16-bit bus sometimes occur when there is no space in the prefetch queue to store the fetched opcodes. This requires another instruction fetch from the same address when space in the prefetch queue opens up.

To the external system, these occurrences appear as duplicate instruction fetches. An estimated 10 percent of all instruction fetches will be "duplicates", while overall bus loading will be approximately 65 to 70 percent, compared to an 8X9X-90 bus loading of approximately 55 to 60 percent. Execution speed is not impacted by a duplicate fetch.

Write Pulse Width: The 8X9XBH 16-bit bus write pulse width is one T_{osc} longer than on the 8X9X-90, thus allowing slower memories and peripherals to be used. In order to widen the **WR** pulse width, the time between the end of **WR** and the next **ALE** was reduced by T_{osc} . Note that the signals **WRL**, **WRH**, and **WR** with an 8-bit bus are still the same width as on -90 parts.

Vpp Replaces VBB: Vpp is the programming pin for EPROM devices. Systems that have this connected through a capacitor to **ANGND** (required on 8X9X-90 parts) do not need to change. **ANGND** must be held nominally at the same potential as **Vss**, and **Vpp** must NOT be connected to **Vcc**. High voltage must NEVER be placed on the **Vpp** pin of a ROM device.

While there is almost no reason to do so, an application should not attempt to execute with the **EA** pin at logic zero and **Vcc** at 5.5 V_{DC} on an 879XBH EPROM device. Additionally, the design should always begin the "out of **RESET**" code execution from the internal EPROM, immediately after the power-on sequence.

Reserved location warning: Intel reserved addresses can not be used by applications which use 8X9XBH internal ROM/EPROM. The data read from a reserved location is not guaranteed, and a write to any reserved location could cause unpredictable results. When attempting to program Intel Reserved addresses, the data must be 0FFFFH to ensure a harmless result.

Intel Reserved locations, when mapped to external memory, must be filled with 0FFFFH to ensure compatibility with future parts.

A positive transition on NMI: The 8X9XBH does not clear the Watchdog Timer. The 8X9X-90 does clear the **WDT** on a positive transition of **NMI**, and both part vector to external address 0000H.

The following is the latest information on upgrading a NMOS 8096 to a CHMOS 80C196.

The chip which is the CHMOS 8096BH replacement is designated the 80C196. The part can be configured to be pin compatible with the 8096, but because of the process change and other enhancements, it may not be plug compatible in some designs. This is to say that you will not be able to arbitrarily swap out a NMOS 8096 and replace it with the 80C196. However, if a few rules are followed the changes required will be almost painless.

80C196 OVERVIEW

First, some background on the 80C196 is needed. The opcode set is a true superset of the 8096, but some enhancements have been made to the peripherals and timings. The crystal is divided by 2 on the 80C196, instead of 3, as on the 8096. This means that the 80C196 running at 8 MHz will have a 250 ns state time, just like an 8096 running at 12 MHz.

An 80C196 running at 8 MHz will emulate an 8096 at 12 MHz except that some of the instructions and peripherals will operate faster. The instructions which will be speeded up include mul, div, interrupt, call, ret, and jumps. The serial port will require a different baud value and the A to D may not run at exactly the same speed. This means that timing loops which measure instruction speed or A to D completion speed may have to be modified. The bus timings, while not nanosecond for nanosecond compatible, will work in most systems.

DESIGN GUIDELINES

1. Do not use undefined register areas for storage or depend on them to return a specific value if it is not stated in the Embedded Controller. Undefined registers and locations on this, or any other, part should be considered off-limits and reserved for development systems, testing or future use.
2. Do not base timing loops on instruction execution times, as some instructions may execute faster on the 80C196 than on the 8096, even when the 80C196 is slowed down to 8 MHz, its 8096 compatible rate. Counter-type loops should be initialized with values that can easily be changed at compile time.

3. Do not base critical timings on interrupt responses, A to D completions, flag settings, etc. This is for the same reason as above; some of these responses may be slightly different from those on the 8096. Timer 1 is provided for critical timings. With an 8 MHz crystal, it will increment every 2 microseconds, just as an 8096 running at 12 MHz.
4. The serial port baud register values should be easily changeable at compile time. Since the serial port is now capable of running at a higher frequency, a different baud rate value will be needed.
5. The circuitry interfacing to the chip should be capable of interfacing to the 80C196. The I/O lines on 80C196 will look a lot like those on the 80C51.
6. The $\overline{\text{BHE}}/\text{WRH}$ signal in eight bit and write strobe mode will go low for odd byte transfers and high for even byte transfers. The $\overline{\text{WR}}/\text{WRL}$ signal will go low for odd byte transfers and high for even byte transfers. Normally, the $\overline{\text{WR}}/\text{WRL}$ signal should go low for odd and even byte transfers since transfers are on the low byte of the data bus.
7. PUSH and POP operations addressed relative to the stack pointer work differently on the 80C196 than on the 8096. On the 8096, the address is calculated based on the un-updated stack pointer value, on the 80C196, the address is calculated based on the updated value. The only operations effected are: PUSH xx[sp], PUSH [sp], PUSH sp, POP xx[sp], POP [sp], POP sp.
8. The V_{PD} pin on the 8X9X parts is now the CDE (Clock Detect Enable) pin on the 80C196. When tied high, CDE enables a clock speed sensor and will reset the part if the Xtal1 frequency drops below a few hundred KHz. While this is perfect for most production boards, it may be desirable to have a jumper option on this function for evaluation boards.

and RAM was picked arbitrarily, and could be reconfigured in various ways, however, this may require slight modifications or additions to the decoder circuitry. This setup has a main page at addresses 8000-07FFFH and upper pages 1-7 at addresses 8000-07FFFH. Note that upper page 0 is the same as the main page. The WR1 and WRH features of the BH part was used to allow for byte writes to RAM. If the -90 part were to be used, additional logic would be necessary to generate these signals from WR and BHE.

The RAM chips utilized were NEC WD63256-15 32K x 8 static rams with an access time of 150ns. The ROMs were Intel 27512 64K x 8 EPROMs with an access time of 300ns. The decoder circuitry used was entirely 1.5 TTL. Using an 8037BH running at 10MHz, there was ample time for address decoding and memory access. Timing analysis showed that 15MHz operation would also be accommodated easily. If slower memories are used, further analysis would be necessary. Also, it would be possible to switch to 2 TTL to greatly decrease the decoding response time.

Software

When using this system there are several things to keep in mind when preparing the software.

Since ASM5d will only allow addresses from 0-FFFFH, it is necessary to use the INST pin of the 8037BH to select the proper page. The main page routine should be coded with addresses from 0-7FFFH, and each of the upper pages should be coded with addresses from 8000-FFFFH. Because linking is not possible, each module should contain a table of constants which defines the symbols used in other modules. These values are easily obtained from the listing file which can be created using xtr in the table the first time. The addresses of the pages in a 27512 after splitting low and high bytes into 2 EPROMs are shown in Figure 1.

RAM				RAM				EPROM				EPROM			
LOCATION				LOCATION				LOCATION				LOCATION			
0000-07FFFH				0000-07FFFH				0000-07FFFH				0000-07FFFH			
PAGE 0				PAGE 0				PAGE 0				PAGE 0			
HIGH BYTES				HIGH BYTES				HIGH BYTES				HIGH BYTES			
PAGE 1				PAGE 1				PAGE 1				PAGE 1			
HIGH BYTES				HIGH BYTES				HIGH BYTES				HIGH BYTES			
PAGE 2				PAGE 2				PAGE 2				PAGE 2			
HIGH BYTES				HIGH BYTES				HIGH BYTES				HIGH BYTES			
PAGE 3				PAGE 3				PAGE 3				PAGE 3			
HIGH BYTES				HIGH BYTES				HIGH BYTES				HIGH BYTES			
PAGE 4				PAGE 4				PAGE 4				PAGE 4			
HIGH BYTES				HIGH BYTES				HIGH BYTES				HIGH BYTES			
PAGE 5				PAGE 5				PAGE 5				PAGE 5			
HIGH BYTES				HIGH BYTES				HIGH BYTES				HIGH BYTES			
PAGE 6				PAGE 6				PAGE 6				PAGE 6			
HIGH BYTES				HIGH BYTES				HIGH BYTES				HIGH BYTES			
PAGE 7				PAGE 7				PAGE 7				PAGE 7			
HIGH BYTES				HIGH BYTES				HIGH BYTES				HIGH BYTES			

DOUG YODER
ECO APPLICATIONS ENGINEER

This Application Brief presents two examples of a page scheme for the 8096 allowing either 32K bytes of total memory or 64K bytes of total memory. Both systems utilize PORT1 as the output for the upper address lines. Because interrupt vectors and other critical sections of code must always be present, addresses 0-7FFFH always refer to the same main page. The PORT1 upper addresses only affect addresses 8000-FFFFH by stepping several 32K pages in and out.

THE 32K SYSTEM

Hardware

The hardware for the 32K system (see Figures 4 & 5, and December 1987) utilizes a 741257 dual 2 to 7 multiplexer. The enable pin of the 741257 is tied to the inverted A15 signal, which is the latched address/data 15 (AD15) signal from the 96. In the way, when A15 is low, the 741257 is disabled and all its outputs are low. Particularly, MA15 is low, which selects the 27512 and deselects the ram. Also, MA15 and MA16 are low, which guarantee that address 0-7FFFH of the 27512 are accessed.

When A15 is high, the 741257 is enabled to pass MA15 - MA17 values. The bank select pin of the 741257 is connected to the INST pin of the 96. When the INST pin is high for a code access, INSTA15 - INSTA17 (PORT15 - PORT17) are used. This allows for the use of separate pages for code and data without having to change the upper address lines each time. Also, it is possible to select a ROM page for a data table, or load a RAM page with executable code downloaded from another source. PORT16 and PORT17 can still be used as I/O ports. If a -90 part were used, the INST pin would need to be latched since it is only valid during the address output on the bus pin.

This system was designed to get the maximum amount of memory with a minimum amount of hardware. The

This Application Brief presents two examples of a paging scheme for the 8096, allowing either 256K bytes of total memory, or 544K bytes of total memory. Both systems utilize PORT1 as the output for the upper address lines. Because Interrupt vectors, and other critical sections of code must always be present, addresses 0-7FFFH always refer to the same main page. The PORT1 upper addresses only affect addresses 8000-FFFFH, by slapping several 32K pages in and out.

THE 256K SYSTEM

Hardware

The hardware for the 256K system (see Figures 4 & 5, an example with 128K ROM and 128K RAM) utilizes a 74LS157 quad 2 to 1 multiplexer. The enable pin of the 74LS157 is tied to the inverted A15 signal, which is the latched addr/data 15 (AD15) signal from the 96. In this way, when A15 is low, the 74LS157 is disabled and all its outputs are low. Particularly, MA17 is low, which selects the 27512 and deselects the rams. Also, MA15 and MA16 are low, which guarantee that addresses 0-7FFFH of the 27512 are accessed.

When A15 is high, the 74LS157 is enabled to pass MA15 - MA17 values. The bank select pin of the 74LS157 is connected to the INST pin of the 96. When the INST pin is high, for a code access, INSTA15 - INSTA17 (PORT1.0 - PORT1.2) are used. When INST is low, for a data read or write, DATAA15 - DATAA17 (PORT1.3 - PORT 1.5) are used. This allows for the use of separate pages for code and data without having to change the upper address lines each time. Also, it is possible to select a ROM page for a data table, or load a RAM page with executable code downloaded from another source. PORT1.6 and PORT1.7 can still be used as I/O ports. If a -90 part were used, the INST pin would need to be latched since it is only valid during the address output on the bus pins.

This system was designed to get the maximum amount of memory with a minimum amount of hardware. The

amount of ROM and RAM was picked arbitrarily, and could be reconfigured in various ways, however, this may require slight modifications or additions to the decoder circuitry. This setup has a main page at addresses 0-7FFFH, and upper pages 1-7 at addresses 8000-FFFFH. Note that upper page 0 is the same as the main page. The WRL and WRH feature of the BH part was used to allow for byte writes to RAM. If the -90 part were to be used, additional logic would be necessary to generate these signals from WR and BHE.

The RAM chips utilized were NEC uPD43256-15 32K x 8 static rams with an access time of 150ns. The ROMs were Intel 27512 64K x 8 EPROMs with an access time of 200ns. The decoder circuitry used was entirely LS TTL. Using an 8097BH running at 10MHz, there was ample time for address decoding and memory access. Timing analysis showed that 12MHz operation would also be accommodated easily. If slower memories are used, further analysis would be necessary. Also, it would be possible to switch to S TTL to greatly decrease the decoding response time.

Software

When using this system there are several things to keep in mind when preparing the software.

Since ASM96 will only allow addresses from 0-FFFFH, it is necessary to generate each page of code in a separate file. These pages should not be linked together, but rather should each be used to program the proper section of the EPROM associated with that page. The main page routine should be coded with addresses from 0-7FFFH, and each of the upper pages should be coded with addresses from 8000-FFFFH. Because linking is not possible, each module should contain a table of constants which defines the symbols used in other modules. These values are easily obtained from the listing file, which can be created using zeros in the table the first time. The addresses of the pages in a 27512 after splitting low and high bytes into 2 EPROMs are shown in Figure 1.

EPROM LOCATION U5		EPROM LOCATION U6		RAM LOCATION U7		RAM LOCATION U8	
0H	MAIN PAGE LOW	0H	MAIN PAGE HIGH	0H	PAGE4 LOW BYTES	0H	PAGE4 HIGH BYTES
3FFFH		3FFFH		3FFFH	PAGE5 LOW BYTES	3FFFH	PAGE5 HIGH BYTES
4000H	PAGE1 LOW BYTES	4000H	PAGE1 HIGH BYTES	4000H		4000H	
7FFFH		7FFFH		7FFFH		7FFFH	
8000H	PAGE2 LOW BYTES	8000H	PAGE2 HIGH BYTES				
BFFFH		BFFFH		0H	PAGE6 LOW BYTES	0H	PAGE6 HIGH BYTES
C000H	PAGE3 LOW BYTES	C000H	PAGE3 HIGH BYTES	3FFFH		3FFFH	
FFFFH		FFFFH		4000H	PAGE7 LOW BYTES	4000H	PAGE7 HIGH BYTES
				7FFFH		7FFFH	

Figure 1. The Current System

EPROM LOCATION U5		EPROM LOCATION U6		EPROM LOCATION U7		EPROM LOCATION U8	
0H	MAIN PAGE LOW	0H	MAIN PAGE HIGH	0H	PAGE4 LOW BYTES	0H	PAGE4 HIGH BYTES
3FFFH		3FFFH		3FFFH		3FFFH	
4000H	PAGE1 LOW BYTES	4000H	PAGE1 HIGH BYTES	4000H	PAGE5 LOW BYTES	4000H	PAGE5 HIGH BYTES
7FFFH		7FFFH		7FFFH		7FFFH	
8000H	PAGE2 LOW BYTES	8000H	PAGE2 HIGH BYTES	8000H	PAGE6 LOW BYTES	8000H	PAGE6 HIGH BYTES
BFFFH		BFFFH		BFFFH		BFFFH	
C000H	PAGE3 LOW BYTES	C000H	PAGE3 HIGH BYTES	C000H	PAGE7 LOW BYTES	C000H	PAGE7 HIGH BYTES
FFFFH		FFFFH		FFFFH		FFFFH	

Figure 2. A System Using all EPROMS and no RAM

All changes to the upper instruction addresses of PORT1 must be made by code located in the main page. A listing of subroutines for use in the main page, and a listing of macros for use in all pages is provided. By invoking one of these macros the programmer can easily transfer from one page to another, or select a new data page. The subroutines should not be called directly, they should be entered by using the appropriate macro. The subroutines should be located at the addresses specified, otherwise the macros must be changed as they are written to call an absolute address in the main page. Also, any hardware changes may render the software inoperative.

Because the WRL-WRH feature of the 96BH is used, the correct Chip Configuration Register value of 0FBH must be loaded into the ROMs at address 2018H. This is done in the main code file with the following statements:

```
CSEG AT 2018H
```

```
CCR: DCB 0FBH ;VALUE FOR CHIP
      CONFIGURATION REGISTER
```

Finally, it is necessary to initialize the DATA address at the start of the program this can be done using the NEW_DATA_PAGE MACRO.

THE 544K SYSTEM

Hardware

The hardware for the 544K system (see Figures 6 & 7, an example with 288K ROM and 256K RAM) has some slight changes from the 256K system.

First, all pins of PORT1 are now in use as address lines. This allows for PORT1 to select 16 pages of memory, with a different address for instructions or data.

Second, 27128 16K x 8 EPROMS have been added for use as the main code page. In this system, the main page is physically separate from upper page 0. The 27128's are selected by A15 being low. The upper pages of memory are selected when A15 is high which enables the 74LS155 demultiplexer which is used for address decoding. When the 74LS155 is disabled, its outputs are all high, which disables all upper memories. The 74LS157 is enabled all the time, to speed up address decoding, as its outputs do not matter when the 74LS155 is disabled.

Software

All rules for the 256K system apply to the 544K system, except that the main page no longer overlaps page 0. However, because all of PORT1 is now in use, different macros and subroutines must now be used. These have been included also.

THE INST PIN

The instruction pin has been verified to work correctly on the 8X9X- 90, 8X9XBH, and the 80C196. The functionality of the INST pin is as follows.

Instruction Fetches

The INST pin is high during an external memory read indicating the read is an instruction fetch. This includes immediate data reads since the data is embedded in the code.

Data Reads and Writes

The INST is low during an external memory read or write indicating the bus cycle is a data cycle. This would be indirect and indexed instructions which are directed at external memory.

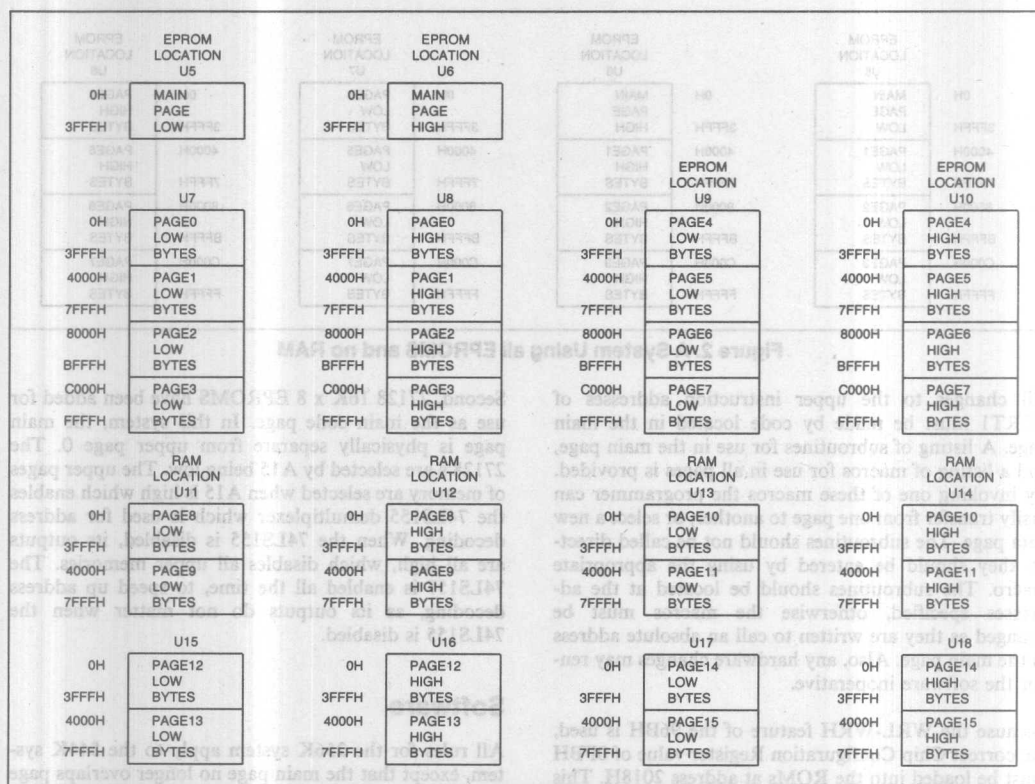
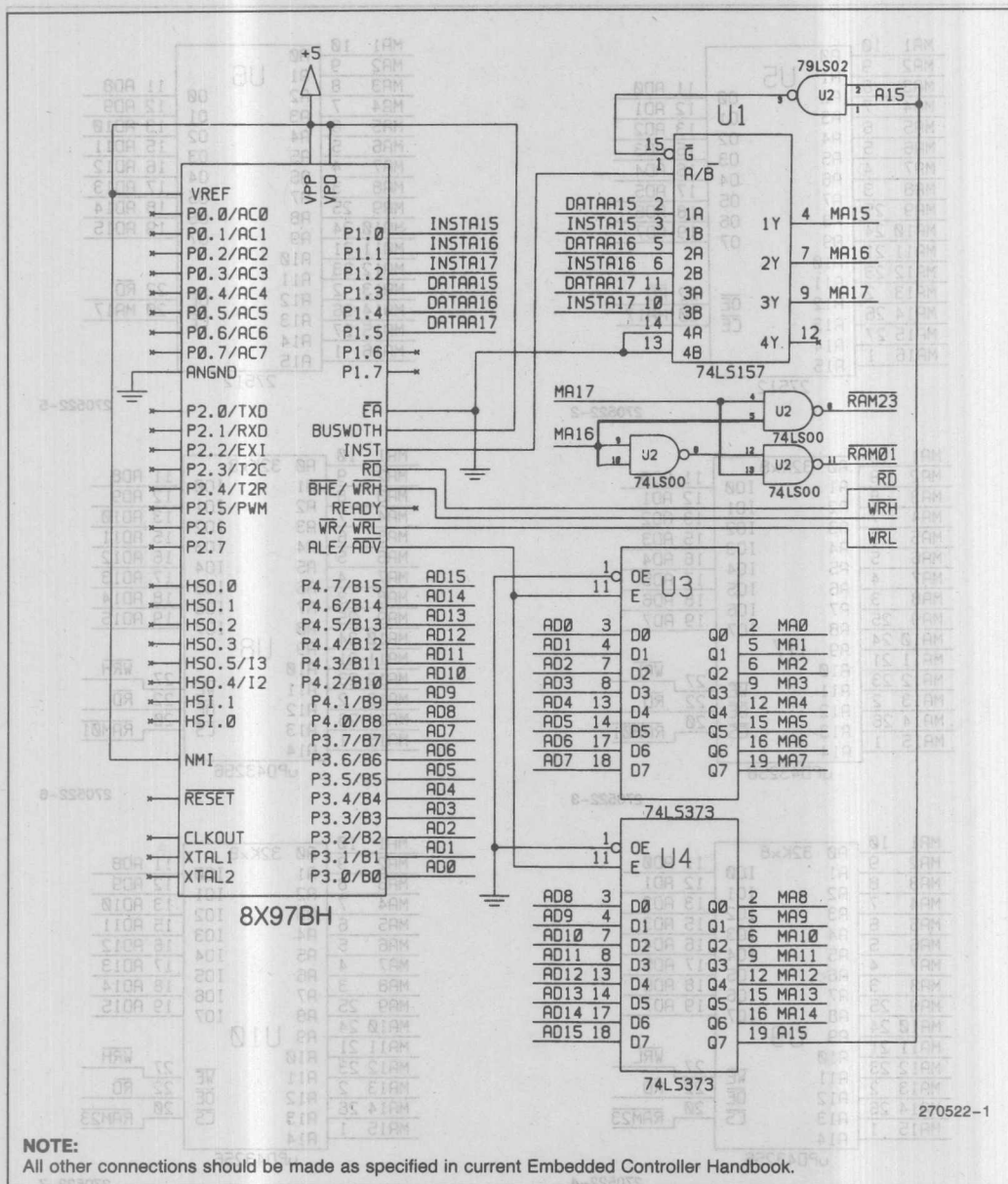


Figure 3. The 544K Memory Map



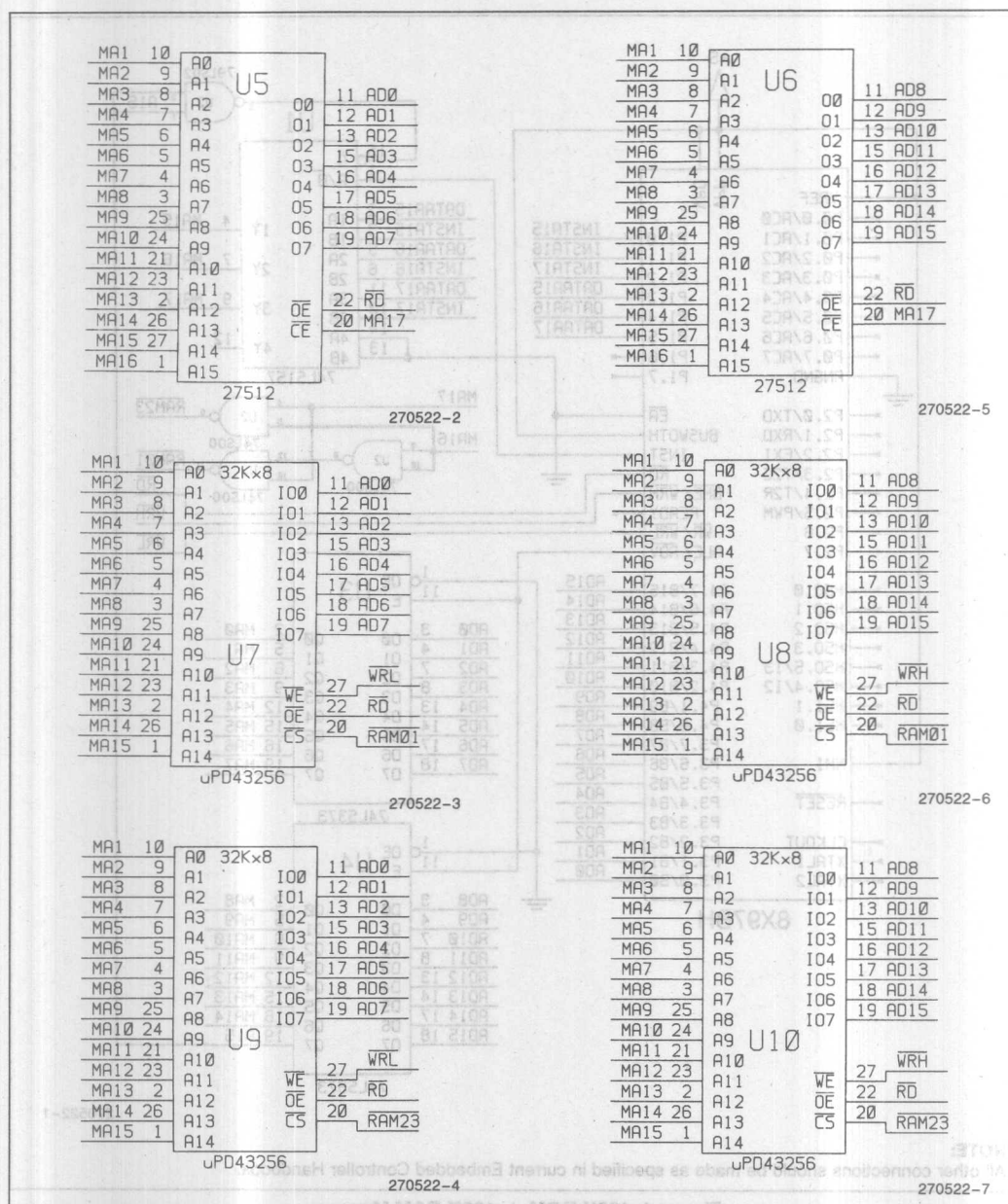


Figure 5. 128K ROM + 128K RAM Memory

;MACROS FOR 256K SYSTEM

;LONG_BRANCH IS INVOKED TO BRANCH FROM ONE PAGE TO ANOTHER.

;ADDRESS MUST HAVE A VALUE FROM 8000H TO FFFFH.

;NEW_PAGE CAN BE AN IMMEDIATE NUMBER OR A REGISTER NUMBER.

```
LONG_BRANCH MACRO ADDRESS, NEW_PAGE
LD CODE_ADDRESS, #ADDRESS ;SET UP CODE_ADDRESS REGISTER
LDB NEW_PAGE_NO, NEW_PAGE ;SET UP NEW_PAGE_NO REGISTER
BR 7FF0H ;BRANCH TO I_P_BRANCH
ENDM
```

;LONG_CALL IS INVOKED TO CALL A SUBROUTINE IN ANOTHER PAGE.

;ADDRESS MUST HAVE A VALUE FROM 8000H TO FFFFH.

;NEW_PAGE CAN BE AN IMMEDIATE NUMBER OR A REGISTER NUMBER.

```
LONG_CALL MACRO ADDRESS, NEW_PAGE
LD CODE_ADDRESS, #ADDRESS ;SET UP CODE_ADDRESS REGISTER
LDB NEW_PAGE_NO, NEW_PAGE ;SET UP NEW_PAGE_NO REGISTER
CALL 7FC0H ;CALL I_P_CALL
ENDM
```

;PUSH_OLD_DATAPAGE IS INVOKED TO INSTALL A NEW DATA PAGE AND SAVE

;THE OLD VALUE ON THE SYSTEM STACK.

;NEW_PAGE CAN BE AN IMMEDIATE NUMBER OR A REGISTER NUMBER.

```
PUSH_OLD_DAPAG MACRO NEW_PAGE
LDB AL, PORT1 ;GET OLD PAGE NUMBER...
PUSH AX ;STORE IT ON THE STACK
LDB AL, NEW_PAGE ;GET NEW DATA PAGE NUMBER...
ANDB AL, #00000111B ;MASK IT...
SHLB AL, #3 ;SHIFT IT TO PROPER POSITION...
ANDB PORT1, #11000111B ;CLEAR THE OLD ONE...
ORB PORT1, AL ;AND LOAD IN NEW ONE
ENDM
```

;POP_OLD_DATAPAGE IS INVOKED TO REINSTALL AN OLD DATA PAGE THAT WAS SAVED
;ON THE SYSTEM STACK BY PUSH_OLD_DATAPAGE.

```
POP_OLD_DAPAG MACRO
POP AX ;RECALL OLD PAGE NUMBER...
ANDB AL, #00111000B ;MASK OLD ONE FOR DATA PAGE...
ANDB PORT1, #11000111B ;CLEAR NEW DATA PAGE...
ORB PORT1, AL ;AND LOAD IN OLD ONE
ENDM
```

;NEW_DATA_PAGE IS INVOKED TO INSTALL A NEW DATA PAGE.

;NEW_PAGE CAN BE AN IMMEDIATE NUMBER OR A REGISTER NUMBER.

```
NEW_DATA_PAGE MACRO NEW_PAGE
LDB AL, NEW_PAGE ;GET NEW DATA PAGE NUMBER...
ANDB AL, #00000111B ;MASK IT...
SHLB AL, #3 ;SHIFT IT TO PROPER POSITION...
ANDB PORT1, #11000111B ;CLEAR THE OLD ONE...
ORB PORT1, AL ;AND LOAD IN NEW ONE
ENDM
```

;SUBROUTINES FOR 256K SYSTEM

CSEG AT 7FC0H

;SUBROUTINE: I_P_CALL

; THIS SUBROUTINE ALLOWS FOR THE CALLING OF SUBROUTINES LOCATED IN
 ; A DIFFERENT PAGE OF MEMORY.
 ;
 ; PARAMETERS: CODE_ADDRESS, NEW_PAGE_NO
 ; SUBROUTINES: ANY THAT ARE REQUESTED.

I_P_CALL: LDB AL, PORT1 ;GET OLD PAGE NUMBER...
 PUSH AX ;STORE IT ON THE STACK
 ANDB PORT1, #11111000B ;CLEAR OLD INST PAGE...
 ANDB NEW_PAGE_NO, #00000111B ;MASK NEW ONE...
 ORB PORT1, NEW_PAGE_NO ;AND LOAD IT IN
 PUSH #I_P_RETURN ;SAVE RETURN ADDRESS...
 BR [CODE_ADDRESS] ;CALL REQUESTED ROUTINE

I_P_RETURN: POP AX ;RECALL OLD PAGE NUMBER...
 ANDB PORT1, #11111000B ;CLEAR NEW INST PAGE...
 ANDB AL, #00000111B ;MASK OLD ONE...
 ORB PORT1, AL ;AND LOAD IT IN
 RET ;RETURN TO CALLING ROUTINE

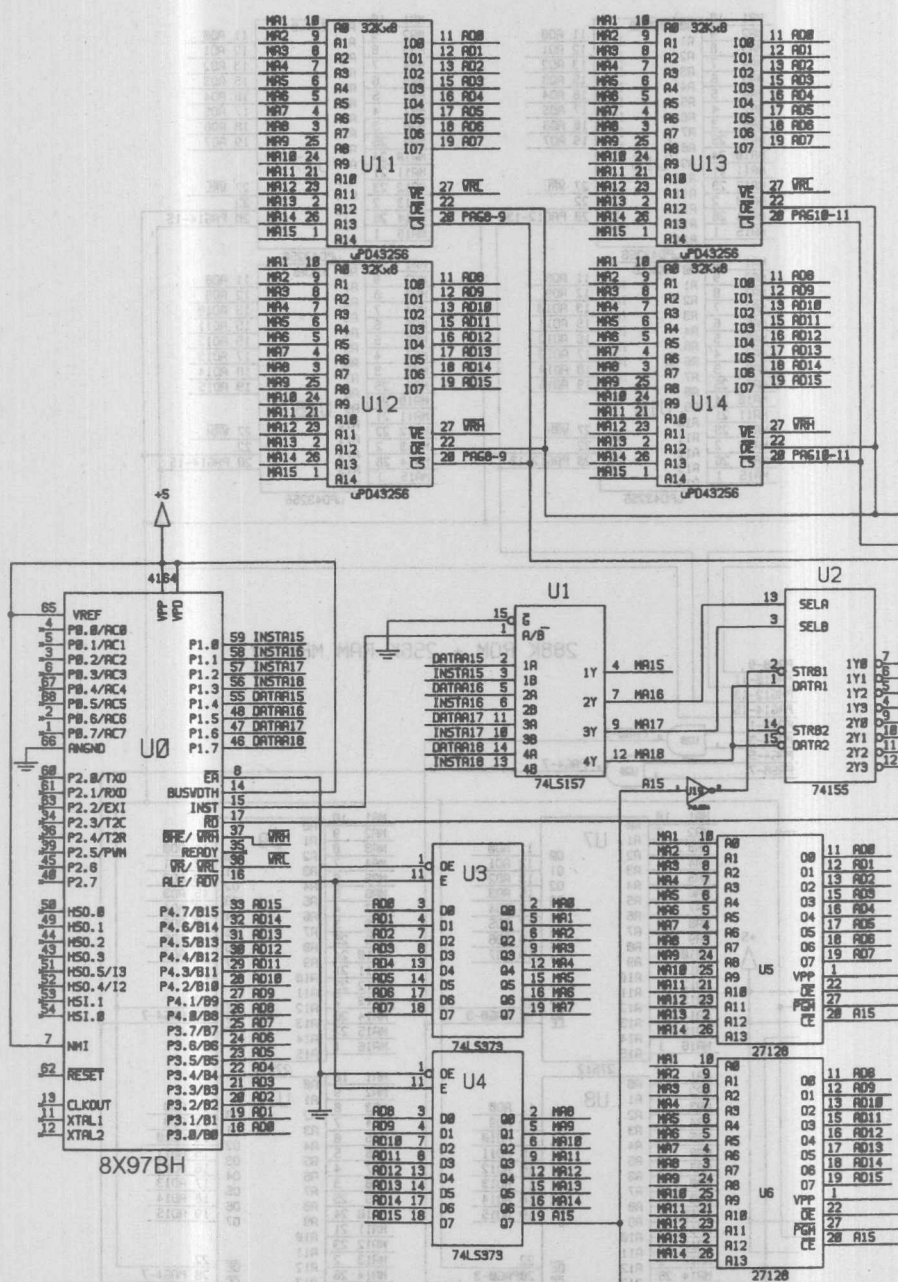
CSEG AT 7FF0H

;SUBROUTINE: I_P_BRANCH

; THIS SUBROUTINE ALLOWS FOR BRANCHING TO LOCATIONS IN A DIFFERENT
 ; PAGE OF MEMORY.
 ;
 ; PARAMETERS: CODE_ADDRESS, NEW_PAGE_NO
 ; SUBROUTINES: NONE

I_P_BRANCH: ANDB PORT1, #11111000B ;CLEAR OLD INST PAGE...
 ANDB NEW_PAGE_NO, #00000111B ;MASK NEW ONE...
 ORB PORT1, NEW_PAGE_NO ;AND LOAD IT IN
 BR [CODE_ADDRESS] ;BRANCH TO REQUESTED ROUTINE

ROUTINE



NOTE:

All other connections should be made as specified in current Embedded Controller Handbook.

Figure 6. 288K ROM + 256K RAM Memory

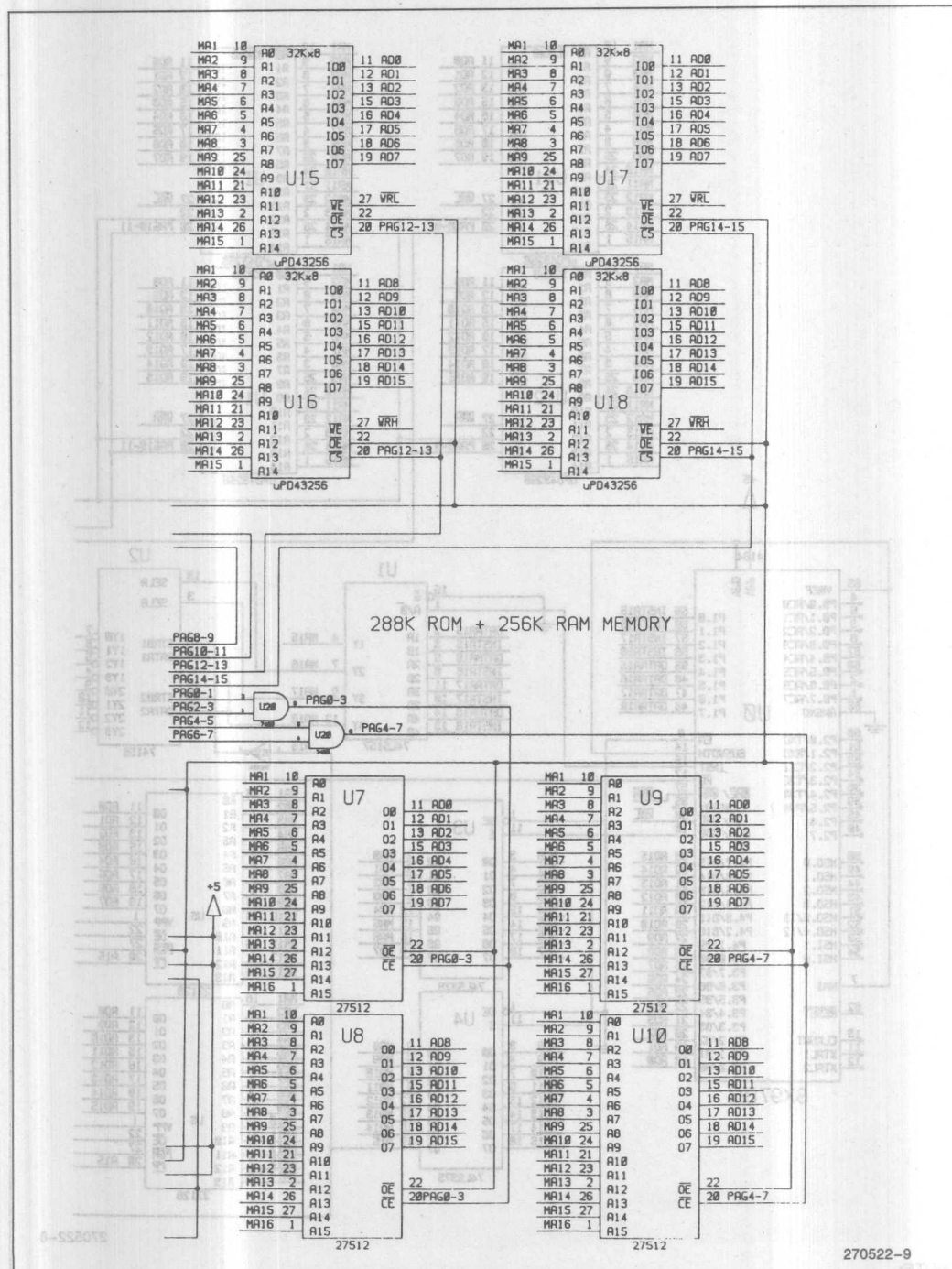


Figure 7. 288K ROM + 256K RAM Memory

;MACROS FOR 544K SYSTEM

;LONG_BRANCH IS INVOKED TO BRANCH FROM ONE PAGE TO ANOTHER.

;ADDRESS MUST HAVE A VALUE FROM 8000H TO FFFFH.

;NEW_PAGE CAN BE AN IMMEDIATE NUMBER OR A REGISTER NUMBER.

```
LONG_BRANCH MACRO ADDRESS, NEW_PAGE
    LD      CODE_ADDRESS, #ADDRESS ;SET UP CODE_ADDRESS REGISTER
    LDB     NEW_PAGE_NO, NEW_PAGE ;SET UP NEW_PAGE_NO REGISTER
    BR      7FF0H ;BRANCH TO I_P_BRANCH
ENDM
```

LONG_CALL IS INVOKED TO CALL A SUBROUTINE IN ANOTHER PAGE.

;ADDRESS MUST HAVE A VALUE FROM 8000H TO FFFFH.

;NEW_PAGE CAN BE AN IMMEDIATE NUMBER OR A REGISTER NUMBER.

```
LONG_CALL MACRO ADDRESS, NEW_PAGE
    LD      CODE_ADDRESS, #ADDRESS ;SET UP CODE_ADDRESS REGISTER
    LDB     NEW_PAGE_NO, NEW_PAGE ;SET UP NEW_PAGE_NO REGISTER
    CALL    7FC0H ;CALL I_P_CALL
ENDM
```

;PUSH_OLD_DATAPAGE IS INVOKED TO INSTALL A NEW DATA PAGE AND SAVE THE OLD
VALUE ON THE SYSTEM STACK.

;NEW_PAGE CAN BE AN IMMEDIATE NUMBER OR A REGISTER NUMBER.

```
PUSH_OLD_DAPAG MACRO NEW_PAGE
    LDB     AL, PORT1 ;GET OLD PAGE NUMBER...
    PUSH    AX ;STORE IT ON THE STACK
    LDB     AL, NEW_PAGE ;GET NEW DATA PAGE NUMBER...
    SHLB    AL, #4 ;SHIFT IT TO PROPER POSITION...
    ANDB    PORT1, #00001111B ;CLEAR THE OLD ONE...
    ORB     PORT1, AL ;AND LOAD IN NEW ONE
ENDM
```

;POP_OLD_DATAPAGE IS INVOKED TO REINSTALL AN OLD DATA PAGE THAT WAS SAVED
ON THE SYSTEM STACK BY PUSH_OLD_DATAPAGE.

```
POP_OLD_DAPAG MACRO
    POP     AX ;RECALL OLD PAGE NUMBER...
    ANDB    AL, #11110000B ;MASK OLD ONE FOR DATA PAGE...
    ANDB    PORT1, #00001111B ;CLEAR NEW DATA PAGE...
    ORB     PORT1, AL ;AND LOAD IN OLD ONE
ENDM
```

;NEW_DATA_PAGE IS INVOKED TO INSTALL A NEW DATA PAGE.

;NEW_PAGE CAN BE AN IMMEDIATE NUMBER OR A REGISTER NUMBER.

```
NEW_DATA_PAGE MACRO NEW_PAGE
    LDB     AL, NEW_PAGE ;GET NEW DATA PAGE NUMBER...
    SHLB    AL, #4 ;SHIFT IT TO PROPER POSITION...
    ANDB    PORT1, #00001111B ;CLEAR THE OLD ONE...
    ORB     PORT1, AL ;AND LOAD IN NEW ONE
ENDM
```

;SUBROUTINES FOR 544K SYSTEM

CSEG AT 7FC0H

;SUBROUTINE: I_P_CALL

; THIS SUBROUTINE ALLOWS FOR THE CALLING OF SUBROUTINES LOCATED IN
; A DIFFERENT PAGE OF MEMORY.; PARAMETERS: CODE_ADDRESS, NEW_PAGE_NO
; SUBROUTINES: ANY THAT ARE REQUESTED.

```

I_P_CALL:    LDB     AL, PORT1          ;GET OLD PAGE NUMBER...
             PUSH    AX                ;STORE IT ON THE STACK
             ANDB    PORT1, #11110000B ;CLEAR OLD INST PAGE...
             ANDB    NEW_PAGE_NO, #00001111B ;MASK NEW ONE...
             ORB     PORT1, NEW_PAGE_NO ;AND LOAD IT IN
             PUSH    #I_P_RETURN       ;SAVE RETURN ADDRESS...
             BR      [CODE_ADDRESS]    ;CALL REQUESTED ROUTINE

I_P_RETURN:  POP     AX                ;RECALL OLD PAGE NUMBER...
             ANDB    PORT1, #11110000B ;CLEAR NEW INST PAGE...
             ANDB    AL, #00001111B    ;MASK OLD ONE...
             ORB     PORT1, AL         ;AND LOAD IT IN
             RET                     ;RETURN TO CALLING ROUTINE

```

CSEG AT 7FF0H

;SUBROUTINE: I_P_BRANCH

; THIS SUBROUTINE ALLOWS FOR BRANCHING TO LOCATIONS IN A DIFFERENT
; PAGE OF MEMORY.; PARAMETERS: CODE_ADDRESS, NEW_PAGE_NO
; SUBROUTINES: NONE

```

I_P_BRANCH:  ANDB    PORT1, #11110000B ;CLEAR OLD INST PAGE...
             ANDB    NEW_PAGE_NO, #00001111B ;MASK NEW ONE...
             ORB     PORT1, NEW_PAGE_NO ;AND LOAD IT IN
             BR      [CODE_ADDRESS]    ;BRANCH TO REQUESTED ROUTINE

```


The largest positive integer you can represent with a 32-bit two's complement number is 07FFFFFFF, or 2,147,483,647. The square root of this number is 00046F, or 46,340. The largest square root that we can generate from a 32-bit radicand can be represented in 16-bits. If we are careful in picking our initial Xold, we can do all of the divisions with the 32 by 16 divide instruction we have available. Picking the largest possible 16-bit number (0FFFF) will always work although it may slow the calculation down by returning too many iterations to arrive at the correct result. The algorithm below takes a slightly more intelligent approach. It uses the normalize instruction to figure out how many leading zeros the 32-bit radicand has and picks an initial Xold based on this information. If there are 16 or more leading zeros then the radicand is less than 16 bits so an initial Xold of 0FFF is chosen. If the radicand is more than 16 bits then the initial Xold is calculated by shifting the value 0FFFF by half as many places as there were leading zeros in the radicand. To give credit where credit is due, I first saw this "trick" in the January 1986 issue of Dr. Dobbs's Journal in a letter from Michael Part of McGill University.

The routine was timed in a 12.0 MHz 8096 as it calculated the square roots of all positive 32-bit numbers, the following numbers include the CALL and return sequence and were measured using Timer 1 of the 8096.

Minimum Execution Time: 24 microseconds

102 microseconds

The program module which follows is part of a collection of routines which perform integer and real arithmetic on a software implemented tagged stack. The top element of the stack is called TOS and is in fixed locations in the register file. Since the square root operation only involves TOS, further details of the stack structure are not shown.

Comments

Integer Square Root Routine for the 8096

LIONEL SMITH
ECO APPLICATIONS ENGINEER

This Application Brief presents an example of calculating the square root of a 32-bit signed integer.

Theory

Newton showed that the square root can be calculated by repeating the approximation:

$$X_{new} = (R/Xold + Xold)/2; Xold = X_{new}$$

where R is the radicand

Xold is the current approximation of the square root

Xnew is the new approximation

December 1987

As you can see, a common technique for finding whether or not you like the answer is to loop on the approximation until Xnew stops changing. If you are dealing with real (floating point) numbers, this technique can sometimes get you in trouble because it's possible to hang up in the loop with Xnew alternating between two values. This is not the case with integers. As an example of how it all works, consider taking the square root of 17 with an initial guess (Xold) of 1:

$$X_{new} = (37/1 + 1)/2 = 19; Xold = 19$$

$$X_{new} = (37/19 + 19)/2 = 10; Xold = 10$$

$$X_{new} = (37/10 + 10)/2 = 6; Xold = 6$$

$$X_{new} = (37/6 + 6)/2 = 3; Xold = 3$$

Note that in integer arithmetic the remainder is ignored and the square root of a number is floored (i.e. the square root is the largest integer which, when multiplied by itself, is less than or equal to the radicand).

Practice

The only significant problem in implementing the square root calculation using this algorithm is that the division of R by Xold could easily be a 32 by 32 divide instruction. This is ok if you happen to have a 32 by 32 divide instruction, but most 16-bit machines (including the 8096) only provide a 32 by 16 divide. However, a little bit of creative laziness will allow us to replace by using the 32 by 16 divide on the 8096.

This Application Brief presents an example of calculating the square root of a 32-bit signed integer.

Theory

Newton showed that the square root can be calculated by repeating the approximation:

$$X_{\text{new}} = (R/X_{\text{old}} + X_{\text{old}})/2; X_{\text{old}} = X_{\text{new}}$$

where: R is the radicand

Xold is the current approximation of the square root

Xnew is the new approximation

until you get an answer you like. A common technique for deciding whether or not you like the answer is to loop on the approximation until Xnew stops changing. If you are dealing with real (floating point) numbers this technique can sometimes get you in trouble because it's possible to hang up in the loop with Xnew alternating between two values. This is not the case with integers. As an example of how it all works, consider taking the square root of 37 with an initial guess (Xold) of 1:

$$X_{\text{new}} = (37/1 + 1)/2 = 19; X_{\text{old}} = 19$$

$$X_{\text{new}} = (37/19 + 19)/2 = 10; X_{\text{old}} = 10$$

$$X_{\text{new}} = (37/10 + 10)/2 = 6; X_{\text{old}} = 6$$

$$X_{\text{new}} = (37/6 + 6)/2 = 6; X_{\text{old}} = 6 - \text{done!}$$

Note that in integer arithmetic the remainder of a division is ignored and the square root of a number is floored (i.e. the square root is the largest integer which, when multiplied by itself, is less than or equal to the radicand).

Practice

The only significant problem in implementing the square root calculation using this algorithm is that the division of R by Xold could easily be a 32 by 32 divide if R is a 32 bit integer. This is ok if you happen to have a 32 by 32 divide instruction, but most 16-bit machines (including the 8096) only provide a 32 by 16 divide. However, a little bit of creative laziness will allow us to squeeze by using the 32 by 16 bit divide on the 8096.

The largest positive integer you can represent with a 32-bit two's complement number is 07fff\$ffffh, or 2,147,483,647. The square root of this number is 0b504h, or 46,340. The largest square root that we can generate from a 32-bit radicand can be represented in 16-bits. If we are careful in picking our initial Xold we can do all of the divisions with the 32 by 16 divide instruction we have available. Picking the largest possible 16-bit number (0ffffh) will always work although it may slow the calculation down by requiring too many iterations to arrive at the correct result. The algorithm below takes a slightly more intelligent approach. It uses the normalize instruction to figure out how many leading zeros the 32-bit radicand has and picks an initial Xold based on this information. If there are 16 or more leading zeros then the radicand is less than 16 bits so an initial Xold of 0fffh is chosen. If the radicand is more than 16 bits then the initial Xold is calculated by shifting the value 0ffffh by half as many places as there were leading zeros in the radicand. To give credit where credit is due, I first saw this 'trick' in the January 1986 issue of Dr. Dobbs's Journal in a letter from Michael Barr of McGill University.

The routine was timed in a 12.0 Mhz 8096 as it calculated the square roots of all positive 32-bit numbers, the following numbers include the CALL and return sequence and were measured using Timer 1 of the 8096.

Minimum Execution Time:	24 microseconds
Maximum Execution Time:	236 microseconds
Average Execution Time:	102 microseconds

Comments

The program module which follows is part of a collection of routines which perform integer and real arithmetic on a software implemented tagged stack. The top element of the stack is call TOS and is in fixed locations in the register file. Since the square root operation only involves TOS, further details of the stack structure are not shown.

```

MCS-96 MACRO ASSEMBLER  SQRT                                05/12/86 10:44:30 PAGE 2-1
DOS MCS-96 MACRO ASSEMBLER, V1.1
SOURCE FILE: ROOT2.A96
OBJECT FILE: ROOT2.OBJ
CONTROLS SPECIFIED IN INVOCATION COMMAND: NOSB
ERR LOC OBJECT      LINE      SOURCE STATEMENT
0019 10 isqrt_id     equ      19h
0000 14 overflow      equ      00h
0001 15 paramerr      equ      01h
0002 16 invalid_input equ      02h
001C 18 oseg at lch
001C 20 ax:          dsw 1
001C 21 al equ ax:byte
001D 22 ah equ (ax+1):byte
001E 23 dx:          dsw 1
0020 24 cx:          dsw 1
0022 25 bx:          dsw 1
0018 26 sp          equ 18h:word
0030 29 oseg at 30h
0030 31 qstk_reg:
0030 32 dsl 1
0030 33 next equ qstk_reg:word
0032 34 tos_tag equ (qstk_reg+2):word
0034 35 tos_value:
0034 36 dsl 1
0000 38 cseg
0000 39 ;
0000 40 bl macro param
0000 41 bnc param
0000 42 endm
0000 43
0000 44 bhe macro param
0000 45 bc param
0000 46 endm
0000 47 $reject

```

```

MCS-96 MACRO ASSEMBLER SQRT                                05/12/86 10:44:30 PAGE 2
ERR LOC OBJECT LINE SOURCE STATEMENT
0000 48 cseg
49 ;
50 ;
51 qstk_isqrt:
52 ; Takes the square root of the long integer in TOS
53 ; TOS -> Top of argument stack
54 ; 1TOS - 1SQRT(TOS)
55 ;
0020 56 Xold set cx
0000 A0341C 57 ld ax,tos_value
0003 A0361E 58 ld dx,(tos_value+2)
0006 371F07 59 bbc (dx+1),7,qsi05 ; if (TOS < 0)
0009 C90119 60 push # (isqrt_id*256+paramerr)
000C EF0000 E 61 call interr ; Call interr.
000F F0 62 ret ; Exit
0010 63 qsi05:
0010 0F221C 64 normal ax, bx
0013 DF3B 65 be qstk_isqrt0
0015 991022 66 cmpb bx,#16 ; if (TOS < 2**16)
0018 DA06 67 ble qsi10
001A 11FF0020 68 ld Xold, #Offh ; Use Offh as first estimate.
001E 200A 69 br qstk_isqrtloop
0020 70 qsi10:
0020 180122 71 shrb bx,#1 ; else
0023 11FFFF20 72 ld Xold, #Offfffh ; Base the first estimate on the
0027 082220 73 shr Xold, bx ; number of leading zeroes in TOS.
002A 74 qstk_isqrtloop:
002A A0341C 75 ld ax,tos_value ; do
002D A0361E 76 ld dx,(tos_value+2) ; if (The divide will overflow)
0030 88201E 77 cmp dx,Xold ; The loop is done.
0035 8C201C 78 bhe qstk_isqrt_done
0038 88201C 79 cmp ax,Xold ; if ( (ax=TOS/Xold) >= Xold)
003D 0122 80 divu ax,Xold ; The loop is done.
003F 641C20 81 cmp ax,Xold ;
0042 A40022 82 bhe qstk_isqrt_done
0045 0C0120 83 clr bx ; Xold=(ax+Xold)/2
0048 27E0 84 add Xold,ax
004A 89 85 addc bx,0
004A A02034 86 shr1 Xold,#1
004D A00036 87 shr1 Xold,#1
0050 F0 88 br qstk_isqrtloop ; while (The loop is not done)
0051 89 qstk_isqrt_done:
0051 90 ld tos_value,Xold ; TOS=00:Xold
0051 91 ld (tos_value+2),0
0051 92 qstk_isqrt0:
0051 93 ret ; Exit
0051 94 end

```

ASSEMBLY COMPLETED. NO ERROR(S) FOUND.



APPLICATION NOTE

AP-406

with digital methods of processing analog signals. This Application Note assists with the first task—understanding of an analog acquisition system.

Designers experienced with analog design, or analog acquisition systems, may find no revelation here. To those unfamiliar with analog acquisition systems, this Ap Note provides a tutorial on the subject and will serve as a handy reference.

Answering the limited number of analog circuit design questions is beyond the scope of this Ap Note. Suffice it to say that the effect placed on the design of analog circuits should increase with a decreasing error budget.

At a minimum, the applications literature of op-amp manufacturers and analog design manuals are a good place to start. Furthermore, the applications literature of monolithic analog acquisition system manufacturers should be consulted since the suggestions presented therein are largely translatable to any A/D system.

This Ap Note is organized in the following sections. The components of an analog acquisition system and the errors associated with each is first explained. Then, interfacing suggestions and ideas for getting more resolution are presented. Finally, a set of appendices provides back-up information: a bibliography, actual converter data and some program listings.

For any user of an MCS-96 analog acquisition system (experienced or not), this document contains very useful information. It should be considered mandatory reading in addition to the latest Embedded Controller Handbook and MCS-96 data sheet for the actual device in use prior to the actual design.

For any user of an MCS-96 analog acquisition system (experienced or not), this document contains very useful information. It should be considered mandatory reading in addition to the latest Embedded Controller Handbook and MCS-96 data sheet for the actual device in use prior to the actual design.

THE MCS-96 ANALOG ACQUISITION PRIMER

INTRODUCTION

As technology advances, embedded control applications continue to reduce chip count and demand microcontrollers with increased features to assist system cost reduction. This is every embedded control application in the physical world, and the physical world is an analog process. It was inevitable that microcontrollers would include integrated analog acquisition capabilities.

December 1987

The introduction of standard microcontrollers in 1978, followed by Intel's 8033 in 1978, opened the door to cost reduction of high volume applications that required analog inputs. The device fit well into applications that needed processing of analog data. But this chip, with its 8-bit CPU, could not perform in high-end applications requiring analog inputs, and applications that had computationally demanding analog tasks.

With the introduction of the MCS-96 family of 16-bit microcontrollers in 1983, the combined CPU and A/D performance became available to greatly reduce the system cost of mid- and high-performance embedded control applications. These are applications which require analog inputs, but which can be implemented with 16-bit microcontrollers. These are applications which require analog inputs, but which can be implemented with 16-bit microcontrollers. These are applications which require analog inputs, but which can be implemented with 16-bit microcontrollers.

To take full advantage of this new level of integration, digital designers must become familiar with analog acquisition, and analog designers must become familiar

MCS[®]-96 Analog Acquisition Primer

DAVID P. RYAN
INTEL CORPORATION

THE MCS®-96 ANALOG ACQUISITION PRIMER

INTRODUCTION

As technology advances, embedded control applications continue to reduce chip-count and demand microcontrollers with increased features to assist system-cost reduction. Since every embedded control application interfaces with the physical world, and the physical world is an analog process, it was inevitable that microcontrollers would include integrated analog acquisition capabilities.

The first such integration of standard microcontroller and A/D converter occurred on Intel's 8022 in 1978. This opened the door to cost reduction of high volume applications that required analog inputs. The device fit well into applications that needed processing of analog data. But this chip, with its 8-bit CPU, could not perform in high-end applications requiring analog inputs, or in applications that had computationally demanding analog tasks.

With the introduction of the MCS®-96 family of 16-bit microcontrollers in 1982, the combined CPU and A/D performance became available to greatly reduce the system cost of mid- and high-performance embedded control applications. These are applications which were customarily implemented with 16-bit microprocessor chip-sets teamed with analog acquisition chip sets.

There are less obvious avenues for system cost reduction when a 16-bit CPU is teamed with an on-chip analog acquisition system. For example, closed-loop servo control had been implemented almost exclusively by using analog methods. When an MCS-96 device is designed into such an application, it is not only replacing a microcontroller or microprocessor, but it also replaces closed-loop analog circuitry which never before came in contact with the digital system.

To take full advantage of this new level of integration, digital designers must become familiar with analog acquisition, and analog designers must become familiar

with digital methods of processing analog signals. This Application Note assists with the first task—understanding of an analog acquisition system.

Designers experienced with analog design, or analog acquisition systems, may find no revelations herein. To those unfamiliar with analog acquisition systems, this Ap Note provides a tutorial on the subject and will serve as a handy reference.

Answering the limitless number of analog circuit design questions is beyond the scope of this Ap Note. Suffice it to say that the effort placed on the design of analog circuits should increase with a decreasing error budget.

At a minimum, the applications literature of op-amp manufacturers and analog design manuals are a good place to start. Furthermore, the applications literature of monolithic analog acquisition system manufacturers should be consulted since the suggestions presented therein are largely transportable to any A/D system.

This Ap Note is organized in the following sections. The components of an analog acquisition system and the errors associated with each is first explained. Then, interfacing suggestions and ideas for getting more resolution are presented. Finally, a set of appendices provides back-up information, a bibliography, actual converter data and some program listings.

The definitions of terms used, and the examples presented, are drawn from the body of applications literature publicly available on the components of an analog acquisition system. There is usually no single meaning for a particular term or specification used to describe analog acquisition. However, there is, in most cases, a generally accepted definition which is most often used. To the extent possible, we have adopted the most used definition. To avoid any ambiguity, Appendix A lists the dictionary of terms as used to refer to the analog acquisition systems of MCS-96 devices.

For any users of an MCS-96 analog acquisition system (experienced or not), this document contains very useful information. It should be considered mandatory reading in addition to the latest Embedded Controller Handbook and MCS-96 data sheet for the actual device in use prior to the actual design.

WHAT IS AN ANALOG ACQUISITION SYSTEM?

An analog acquisition system is a collection of individual units which, when logically configured, form a system capable of converting an analog input to a digital value.

The typical components of an Analog Acquisition Unit (Figure 1) include an Analog-to-Digital Converter (A/D), a Sample-and-Hold (S/H) and an Analog Multiplexer (MUX). The A/D converts the infinitely varying analog voltage present on the S/H into a digital representation for use by the digital system. The S/H is required so a "snapshot" of a changing analog input can be stored for conversion by the A/D. The MUX is used to leverage the investment in the A/D by allowing a large number of isolated analog input channels to use the same converter.

The conversion result of an MCS-96 device is a 10-bit ratiometric representation of the input voltage. This produces a stair-stepped transfer function when the output code is plotted versus input voltage. See Figure 2.

The resulting digital codes can be taken as simple ratiometric information, or they can be used to provide information about absolute voltages or relative voltage changes on the inputs. The more demanding the application is on the A/D converter, the more important it is to fully understand the converter's operation. For simple applications, knowing the absolute error of the converter is sufficient. However, controlling a closed loop with analog inputs necessitates a detailed understanding of an A/D converter's operation and errors.

The errors inherent in an analog-to-digital conversion process are many: quantizing error; zero offset; full-

scale error; differential non-linearity; and non-linearity. These are "transfer function" errors related to the A/D converter. In addition, the S/H and MUX may induce channel dissimilarities and sampling error (described later).

Fortunately, one "Absolute Error" specification is available which describes the sum total of all deviations between the actual conversion process and an ideal converter. The various sub-components of error are, however, important in many applications. These error components are described in Appendix A and in the text below where ideal and actual converters are compared.

A/D Converter

There are at least three well-recognized methods for converting an analog voltage to a digital value—flash, dual slope and successive approximation.

Flash A/Ds are the fastest, and most expensive converters for a given accuracy. Flash converters typically resolve bits of the result in parallel to achieve fast conversions. Flash converter speeds are measured in tens-of-nanoseconds.

Dual slope converters are the slowest, but most accurate. Dual slope conversion is rather insensitive to noise on the input, but conversion times are measured in milliseconds.

Successive approximation converters provide a balanced tradeoff between speed and accuracy. Successive approximation conversion times are measured in tens-of-microseconds, and converter implementations are very economical for a given accuracy.

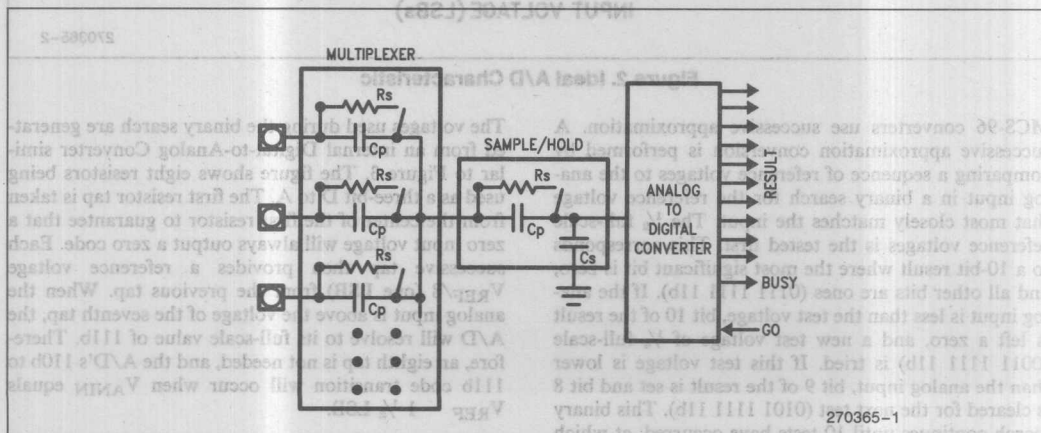


Figure 1: An Analog Acquisition System

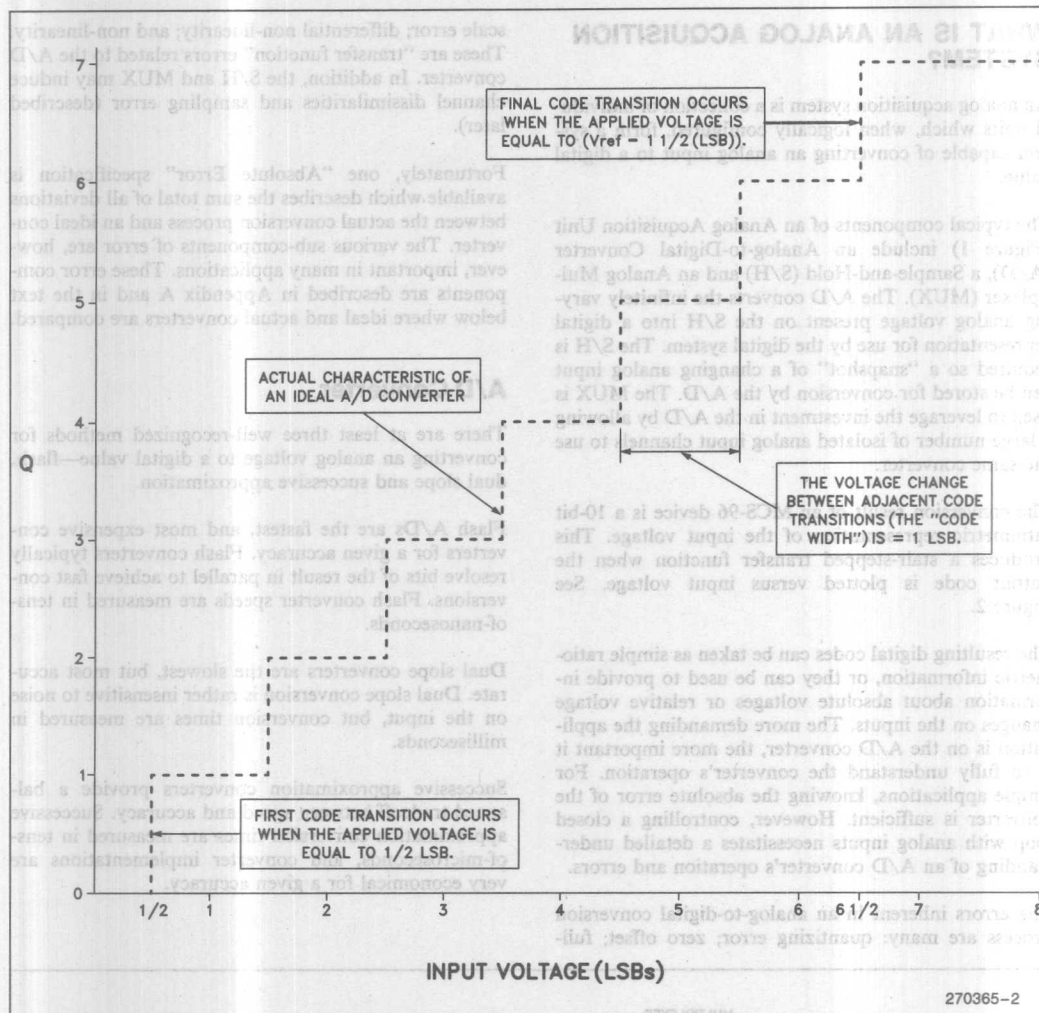


Figure 2. Ideal A/D Characteristic

MCS-96 converters use successive approximation. A successive approximation conversion is performed by comparing a sequence of reference voltages to the analog input in a binary search for the reference voltage that most closely matches the input. The $\frac{1}{2}$ full-scale reference voltages is the tested first. This corresponds to a 10-bit result where the most significant bit is zero, and all other bits are ones (0111 1111 11b). If the analog input is less than the test voltage, bit 10 of the result is left a zero, and a new test voltage of $\frac{1}{4}$ full-scale (0011 1111 11b) is tried. If this test voltage is lower than the analog input, bit 9 of the result is set and bit 8 is cleared for the next test (0101 1111 11b). This binary search continues until 10 tests have occurred, at which time the valid 10-bit conversion result resides in a register where it can be read by software.

The voltages used during the binary search are generated from an internal Digital-to-Analog Converter similar to Figure 3. The figure shows eight resistors being used as a three-bit D to A. The first resistor tap is taken from the center of the first resistor to guarantee that a zero input voltage will always output a zero code. Each successive tap then provides a reference voltage $V_{REF}/8$ (one LSB) from the previous tap. When the analog input is above the voltage of the seventh tap, the A/D will resolve to its full-scale value of 111b. Therefore, an eighth tap is not needed, and the A/D's 110b to 111b code transition will occur when V_{ANIN} equals $V_{REF} - 1\frac{1}{2}$ LSB.

The first error seen in this process is unavoidable, and results from the conversion of a continuous voltage to

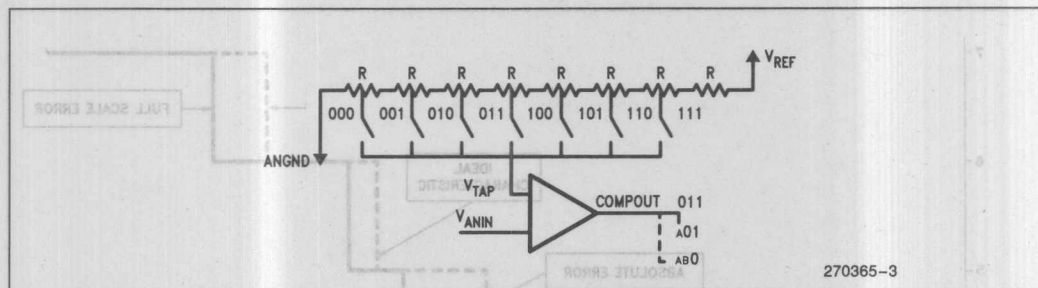


Figure 3. A Three-Bit D-to-A

an integer digital representation. This error is called quantizing error, and is always ± 0.5 LSB. Quantizing error is the only error seen in a perfect A/D converter, and is obviously present in actual converters. Figure 2 shows the transfer function for an ideal 3-bit A/D converter (i.e. the Ideal Characteristic).

Note that in Figure 2 the Ideal Characteristic possesses unique qualities: it's first code transition occurs when the input voltage is 0.5 LSB; it's full-scale code transition occurs when the input voltage equals the full-scale reference minus 1.5 LSB; and it's code widths are all exactly one LSB. These qualities result in a digitization without offset, full-scale or linearity errors. In other words, a perfect conversion.

Figure 4 shows an Actual Characteristic of a hypothetical 3-bit converter which is not perfect. When the Ideal Characteristic is overlaid with the imperfect characteristic, the actual converter is seen to exhibit errors in the location of the first and final code transitions and code widths. The deviation of the first code transition from ideal is called "zero offset". The deviation of the final code transition from ideal is "full-scale error".

The deviation of the code widths from ideal causes two types of errors. Differential Non-Linearity and Non-Linearity. Differential Non-Linearity is a local linearity error measure, whereas Non-Linearity is an overall linearity error measure. For example, Figure 5a shows a transfer function with a large differential non-linearity and a little non-linearity. In contrast, Figure 5b shows a characteristic with small differential errors but a large overall linearity error.

Differential Non-Linearity is the degree to which actual code widths differ from the ideal width. Differential Non-Linearity gives the user a measure of how much the input voltage may have changed in order to produce a one count change in the conversion result.

If the absolute value of an input voltage is less important than the amount that the input changes, the differential non-linearity (DNL) specification of a converter is very important. For example, if the differential non-linearity of a converter is less than ± 0.5 LSB, a one count change in the digital result means that the input voltage changed at most 1.5 LSB (1 LSB ideal ± 0.5 LSB DNL). This is a much more accurate description of the input voltage change than would be available if the differential non-linearity of the converter was not known.

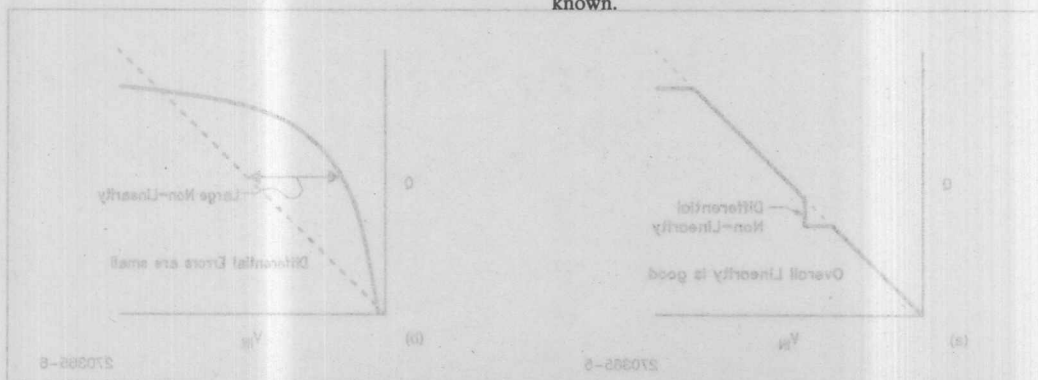


Figure 4. Types of Linearity Errors

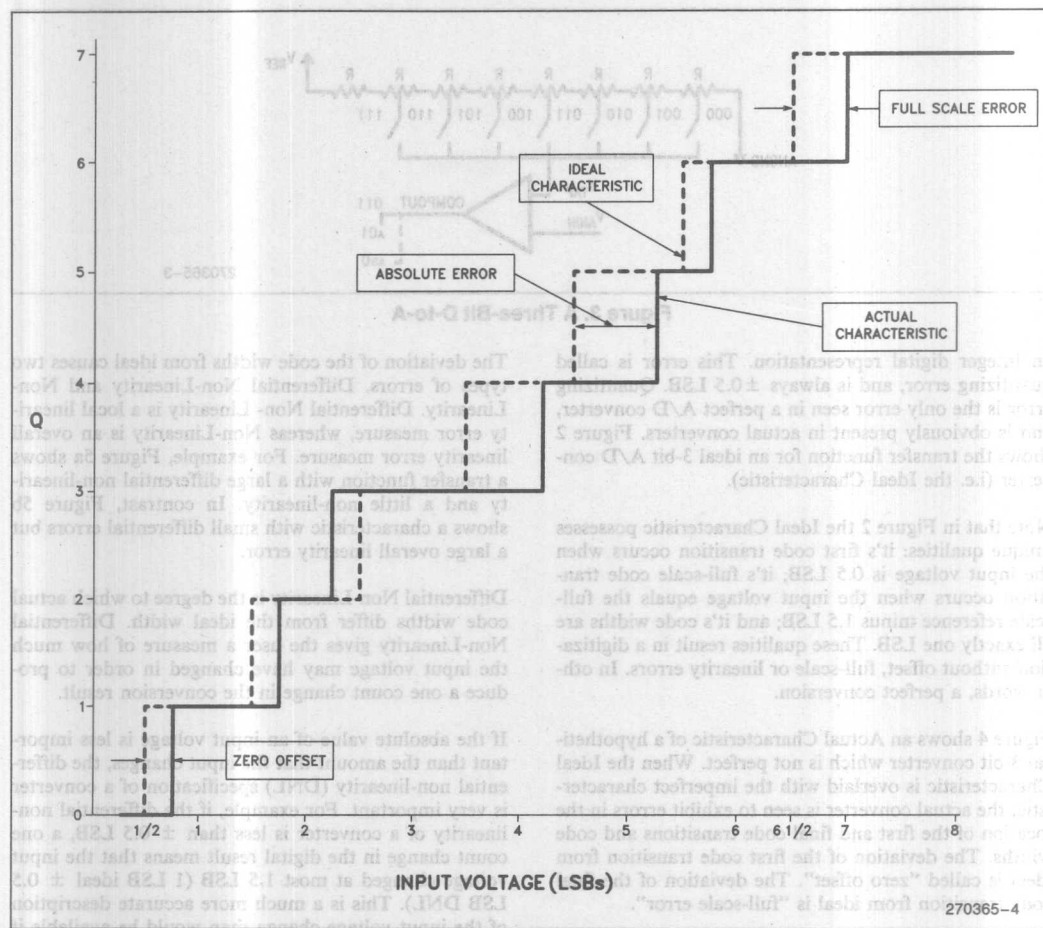


Figure 4. Actual and Ideal Characteristics

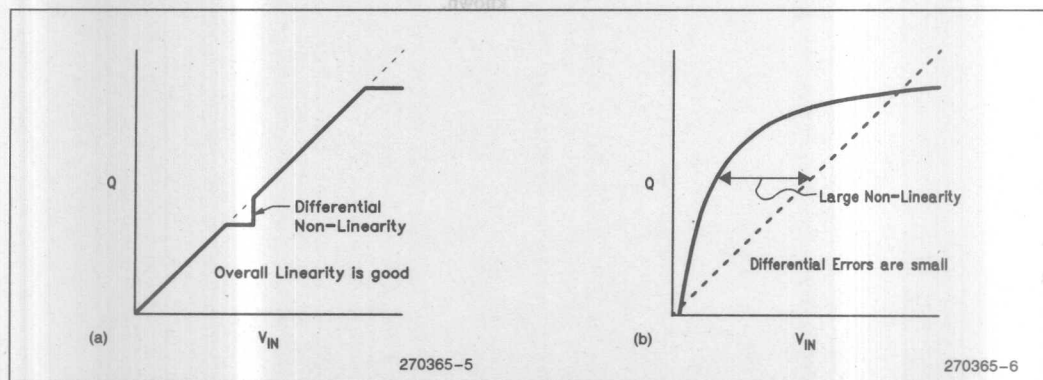


Figure 5. Types of Linearity Errors

Non-Linearity is the worst case deviation of code transitions from the corresponding code transitions of the Ideal Characteristic. Non-Linearity describes how much Differential Non-Linearities could add to produce an overall maximum departure from a linear characteristic.

If the Differential Non-Linearity errors are large enough, it is possible for an A/D converter to miss codes or exhibit non-monotonicity. Neither behavior is desirable in a closed-loop system. A converter has no missed codes if there exists for each output code a unique input voltage range that produces that code only. A converter is monotonic if every subsequent code change represents an input voltage change in the same direction. Figure 6a shows a converter with missed codes. Figure 6b shows a non-monotonic converter.

Differential Non-Linearity and Non-Linearity are quantified by measuring the Terminal Based Linearity Errors. A Terminal Based Characteristic results when an Actual Characteristic is shifted and scaled to eliminate zero offset and full-scale error (see Figure 7). The Terminal Based Characteristic is similar to the Actual Characteristic that would be seen if zero offset and full-scale error were externally trimmed away. In practice, this is done by using input circuits which include gain and offset trimming. (See the Application Hints section for more details.)

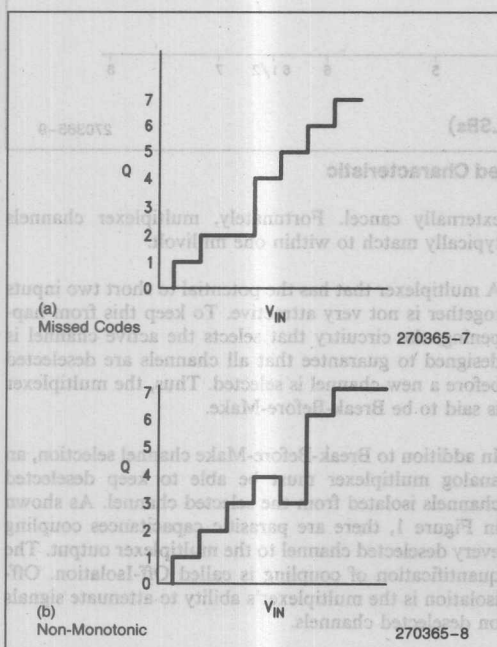


Figure 6. Undesirable Converter Operation

An often overlooked characteristic of A/D converters is that code transitions do not really occur instantaneously at some finite set of input voltages. Specific code transitions can be analyzed by doing repeated conversions around the transition point using a high accuracy input voltage. When this is done, we find that there is actually a range of voltages around code transitions where both the lower and upper codes occur for repeated conversions on the same input voltage.

Figure 8 shows this "repeatability" error. At the lower end of the region of repeatability error the lower code is most prevalent, but the upper code will occur in a small percentage of the conversion attempts. As the input voltage increases slightly, a point is reached where both lower and upper codes occur with 50 percent probability. As the input voltage moves slightly higher, the upper code occurs most often with the lower code showing up in a small percentage of conversions.

The repeatability error is due to the fundamental ability of the comparator in the A/D to resolve very similar voltages. Random noise also contributes to repeatability errors. On MCS-96 devices, the width of the region of repeatability error has been found to be typically 1 mV to 1.25 mV. Since this error is specified, all other errors are specified assuming the code transitions occur at the voltage where adjacent codes are equally likely.

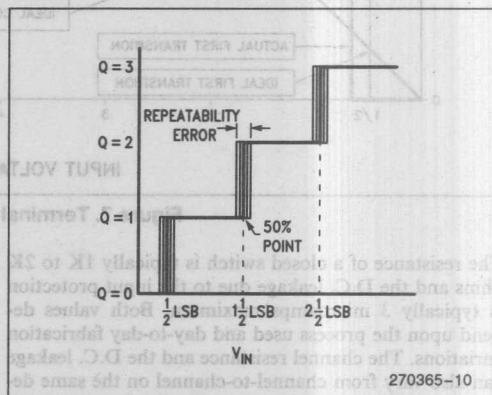


Figure 8. Repeatability Error

The Multiplexer

The eight channel multiplexer is implemented as a collection of eight MOS switches. Only one of eight can be closed at any instant in time. Figure 1 shows the multiplexer with the switches acting as resistors when closed and as small parasitic capacitors when open. The input protection devices on the analog input pins are also considered a part of the multiplexer.

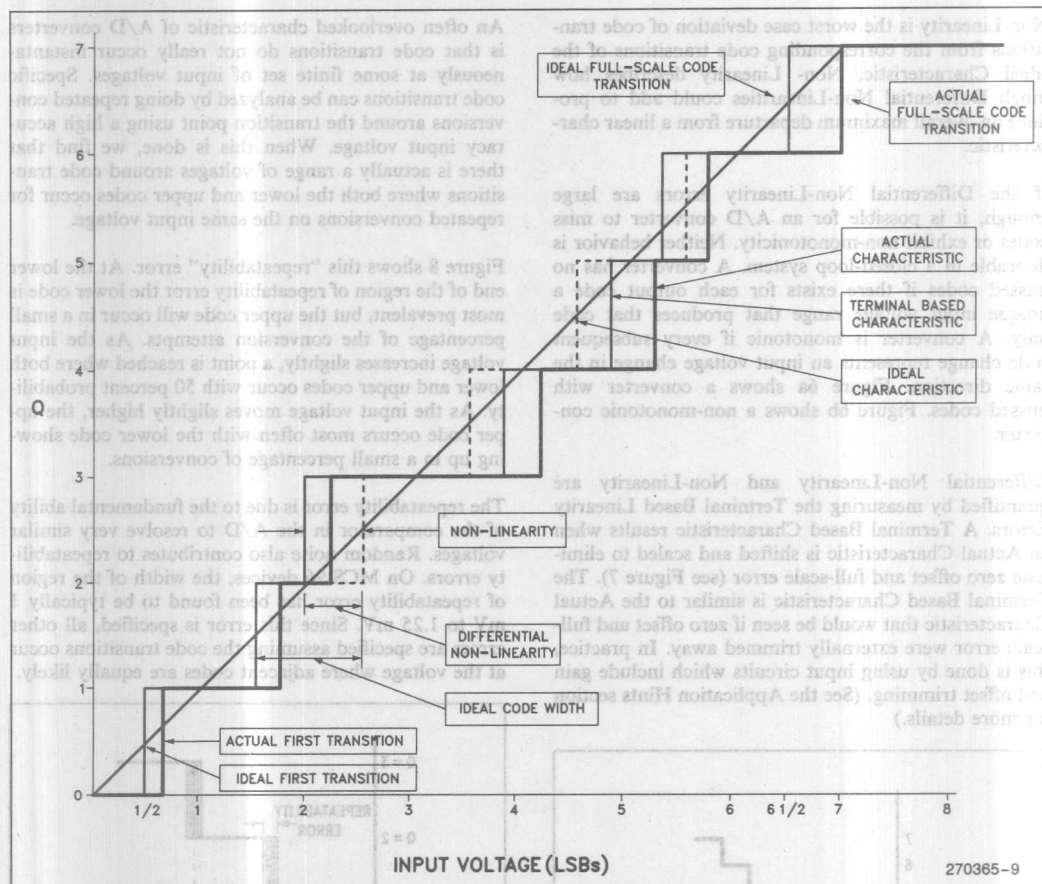


Figure 7. Terminal Based Characteristic

The resistance of a closed switch is typically 1K to 2K ohms and the D.C. leakage due to the input protection is typically 3 microamps maximum. Both values depend upon the process used and day-to-day fabrication variations. The channel resistance and the D.C. leakage can also vary from channel-to-channel on the same device. These variations can be seen in the conversion process and are described by the channel-to-channel matching specification.

Channel-to-channel matching specifies the input voltage differences induced by mismatched elements of the multiplexer. This error is quantified by measuring the difference between the input voltages necessary to cause the same code transition to occur through different multiplexer channels under identical test conditions.

Matching errors are more complex than a simple voltage offset between channels, and thus are difficult to

externally cancel. Fortunately, multiplexer channels typically match to within one millivolt.

A multiplexer that has the potential to short two inputs together is not very attractive. To keep this from happening, the circuitry that selects the active channel is designed to guarantee that all channels are deselected before a new channel is selected. Thus, the multiplexer is said to be Break-Before-Make.

In addition to Break-Before-Make channel selection, an analog multiplexer must be able to keep deselected channels isolated from the selected channel. As shown in Figure 1, there are parasitic capacitances coupling every deselected channel to the multiplexer output. The quantification of coupling is called Off-Isolation. Off-isolation is the multiplexer's ability to attenuate signals on deselected channels.

Sample-and-Hold

The sample-and-hold of an analog acquisition system can be built using an analog switch and a sample capacitor. As with the multiplexer, there is also a parasitic capacitance coupling the switch input to the sample capacitor when the switch is open (Figure 1).

The resistance of the sample-and-hold switch combines with the series resistance of the multiplexer to impede the current necessary to charge the sample capacitor. For example, with a 5K ohm total input resistance from the pin to the 2 pf sample capacitor, the RC time constant is 10 nS ($2 \text{ pf} \times 5\text{K ohms}$).

During the one microsecond that the sample capacitor is connected to the input, 100 time constants elapse (1 microsecond/10 nS). This means that the sample capacitor is 100 percent of the voltage on the input pin ($1 - e^{-100}$), assuming a zero source impedance.

If a source impedance of 2K ohms is assumed, the RC time constant of the sampling process would be 14nS ($7\text{K ohms} \times 2 \text{ pf}$). Thus, 71.4 time constants would pass in one microsecond resulting in the sample capacitor being charged to within 99.9 percent of its final value. Source impedances above 2K ohms would begin to degrade the conversion accuracy due to D.C. leakage (described later).

Figure 9 shows the actual input voltage and the sampled voltage approaching the input voltage. Once the sample-and-hold switch closes, the sample window begins. The sample window extends for four state times and ends with the sample-and-hold switch opening on MCS-96 devices (except 8X9X-90, which is 8 state times and has no sample-hold). Figure 9 also shows the sample delay, which is the delay from the time a start conversion signal is generated to the time a conversion process begins.

It is important to understand the uncertainties associated with the timing of the sample-and-hold. Digital signal processing algorithms rely upon the "spectral purity" of the sampling process. If the sample window jumps around with respect to the start conversion signal, or if the start conversion signal cannot be generated at precise times, consecutive samples of input data will not be equally spaced in time (i.e. sampling will be spectrally impure).

The total number of state times required for a 10-bit conversion varies from one MCS-96 version to the next. Attempting to short-cut the 10-bit conversion process by reading A/D results before the done bit is set may work on some versions of MCS-96 devices, however it is not recommended. Short-cutting is not tested, nor is it guaranteed. Furthermore, it may not work on future MCS-96 devices.

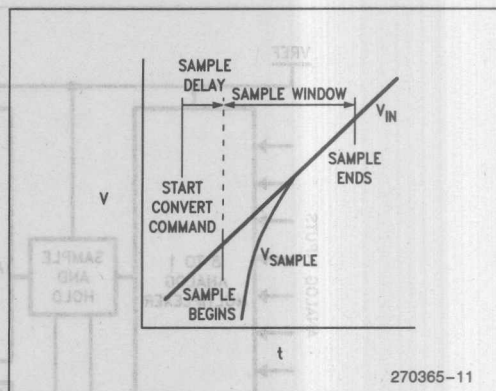


Figure 9. Sample-and-Hold Voltage

To improve the spectral purity of the sampling in digital signal processing applications, sequential MCS-96 start conversion signals can be generated with less than 50 nanoseconds of jitter using the HSO unit. The sample delay and sample time are also a constant number of state times to within 50 nanoseconds each.

Once the sample window closes, it is desired that all further changes on any input channel be isolated from the sample capacitor. The multiplexer's off-isolation is responsible for isolating deselected channels, while the sample-and-hold switch must attenuate changes on the selected channel. This source of error is described as Feedthrough. Feedthrough is quantified as the ability of the sample-and-hold to reject unwanted signals on its input.

Other factors that affect a real A/D Converter system include sensitivity to temperature. Temperature sensitivities are described by the change in typical specifications with a change in temperature.

The MCS®-96 Conversion Sequence

The MCS-96 Analog Acquisition System includes an eight channel analog multiplexer, sample-and-hold circuit and 10-bit analog to digital converter (Figure 10). An MCS-96 device can therefore select one of eight analog inputs, sample-and-hold the input voltage and convert the voltage into a digital value. Each conversion takes 22 microseconds (8097BH), including the time required for the sample-hold (with XTAL1 = 12 MHz). The method of conversion is successive approximation.

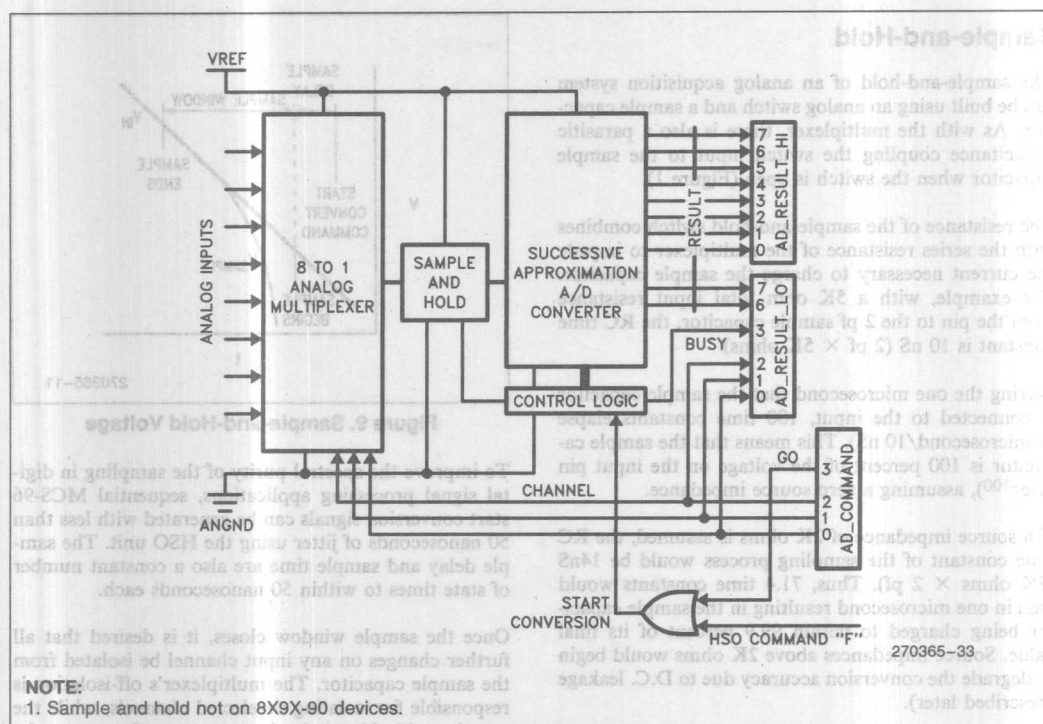


Figure 10. A/D Converter Block Diagram

The conversion process is initiated by the execution of HSO command OFH, or by writing a one to the GO Bit in the A/D Control Register. Either activity causes a start conversion signal to be sent to A/D control logic. If an HSO command was used, the conversion process will begin when Timer 1 increments. This aids applications attempting to approach spectrally pure sampling, since successive samples spaced by equal Timer 1 delays will occur with a variance of about ± 50 ns (assuming a stable clock on XTAL1). However, conversion initiated by writing a one to the ADCON register GO Bit will start within three state times after the instruction has completed execution, resulting in a variance of about $0.75 \mu\text{s}$ (XTAL1 = 12 MHz).

Once the A/D unit receives a start conversion signal, there is a one state time delay before sampling (sample delay) while the successive approximation register is reset and the proper multiplexer channel is selected. After the sample delay, the multiplexer output is connected to the sample capacitor and remains connected for four state times (sample time). After this four state time "sample window" closes, the input to the sample capacitor is disconnected from the multiplexer so that changes on the input pin will not alter the stored charge while

the conversion is in progress. The sample delay and sample time uncertainties are each approximately ± 50 ns, independent of clock speed.

To perform the actual analog-to-digital conversion the MCS-96 implements a successive approximation algorithm. The converter hardware consists of a 256-resistor ladder, a comparator, coupling capacitors and a 10-bit successive approximation register (SAR) with logic that guides the process. The resistor ladder provides 20 mV steps ($V_{\text{REF}} = 5.12\text{V}$), while capacitive coupling is used to create 5 mV steps within the 20 mV ladder voltages. Therefore, 1024 internal reference voltages are available for comparison against the analog input to generate a 10-bit conversion result. Appendix B contains a detailed description of the method used to generate 1024 voltages from a 256-resistor chain.

The total number of state times required for a 10-bit conversion varies from one MCS-96 version to the next. Attempting to short-cycle the 10-bit conversion process by reading A/D results before the done bit is set may work on some versions of MCS-96 devices, however it is not recommended. Short-cycling is not tested, nor is it guaranteed. Furthermore, it may not work on future MCS-96 devices.

APPLICATION HINTS

The analog signals that must be converted by an analog acquisition system vary widely. The analog input may arrive at the controller as a voltage or current. The range may be 0 to 1 volt or ± 30 volts, or some other arbitrary range. The input may be linear, logarithmic, non-linear, or perturbed in some bizarre fashion. Although interfacing to such signals could be considered an art form, some simple suggestions are contained in this section.

Analog Inputs

The external interface circuitry to an analog input is highly dependent upon the application, and can impact converter characteristics. In the external circuit's design, important factors such as input pin leakage, sample capacitor size and multiplexer series resistance from the input pin to the sample capacitor must be considered.

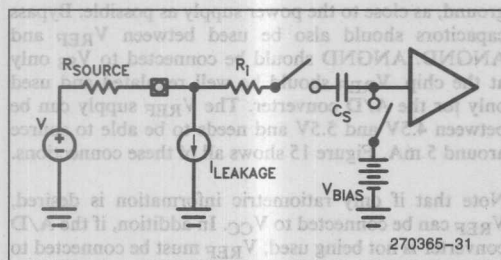


Figure 11. Idealized A/D Sampling Circuitry

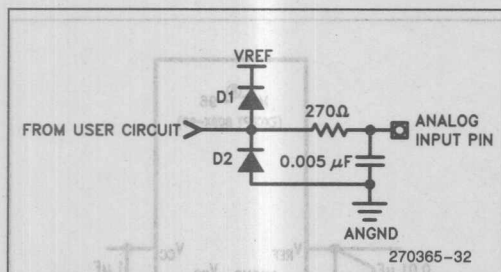


Figure 12. Suggested A/D Input Circuit

For the 8096BH, these factors are idealized in Figure 11. The external input circuit must be able to charge a sample capacitor (C_S) through a series of resistance (R_I) to an accurate voltage given a D.C. leakage (I_L). On the 8096BH, C_S is around 2 pF, R_I is around 5 K Ω and I_L is specified at 3 μ A maximum. In determining the source impedance R_S , V_{BIAS} is not important.

External circuits with source impedances of 1 K Ω or less will be able to maintain an input voltage within a

tolerance of about ± 0.61 LSB ($1.0 \text{ K}\Omega \times 3.0 \mu\text{A} = 3.0 \text{ mV}$) given the D.C. leakage. Source impedances above 2 K Ω can result in an external error of at least one LSB due to the voltage drop caused by the 3 μ A leakage. In addition, source impedances above 25 K Ω may degrade converter accuracy as a result of the internal sample capacitor not being fully charged during the 1 μ s (12 MHz clock) sample window.

Placing an external capacitor on each analog input will reduce the sensitivity to noise, as the capacitor combines with source resistance in the external circuit to form a low-pass filter. In practice, one should include a small series resistance prior to an external low leakage capacitor on the analog input pin and choose the largest capacitor value practical, given the frequency of the signal being converted. This provides a low-pass filter on the input, while the resistor will also limit input current during over-voltage conditions.

Figure 12 shows a simple analog interface circuit based upon the discussion above. The circuit in the figure also provides limited protection against over-voltage conditions on the analog input (limits to 2.6 mA with 270 Ω ($0.7/270$)). The circuit induces leakage from the diodes, which should be kept small.

The wide range of possible analog environments that must be interfaced to, or the existence of stringent accuracy requirements, makes the consideration of alternative input buffer configurations necessary. The most popular input buffer is a single op-amp in the non-inverting or inverting configurations of Figure 13.

In the non-inverting circuit of Figure 13 (a), the analog input is scaled by the buffer gain to output 5 volts when the input is at its maximum positive input. When the buffer input is 0 volts, the output will also be 0 volts.

In the inverting circuit of Figure 13 (b), a reference equal to the maximum possible input voltage is placed on the non-inverting input of the op-amp and the actual analog input is placed on the inverting input. The output voltage of the buffer is then proportional to the deviation of analog input from its maximum possible value. For example, when the analog input equals V_{MAX} , the buffer output will equal 0 zero volts. When the analog input equals its minimum value, the buffer output equals 5 volts. The digital result from the A/D converter might, of course, have to be complemented before being used.

The circuits of Figure 13 show only feedback resistors that set the gain of the buffer. In practice, it will often be necessary to include offset adjustments, gain trimming, temperature or frequency stability compensation, or components to build an active filter.

Figure 14 depicts a generalized non-inverting input buffer that offsets the analog input and scales the input

to a 5 volt range. The course offset is set by the ratio of R_{BIG1} and R_{BIG2} , while offset fine tuning is done by adjusting R_{TRIM} . The course gain is set by the ratio of R_{G1} and R_{G2} while gain trimming is done with R_{GTRIM} .

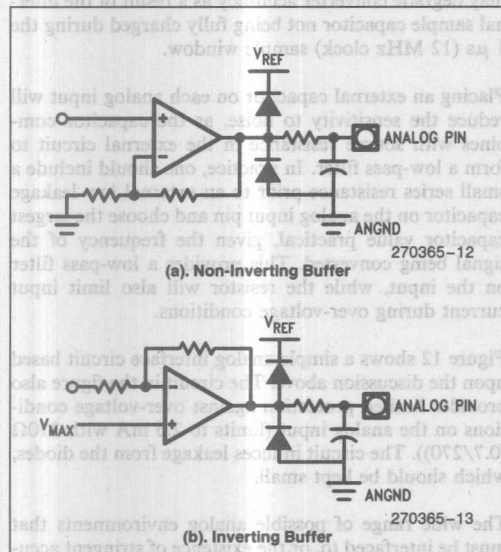


Figure 13

By trimming the offset and gain, not only can external component errors be zeroed out, but the offset and full scale error of the A/D converter can be nulled.

The procedure for nulling offset and gain is simple. First, a voltage is applied to V_{IN} which corresponds to the ideal first code transition of the A/D. R_{TRIM} is adjusted so that 50 percent of the conversion results are 0 while 50 percent are 1. Second, a voltage is applied to V_{IN} which corresponds to the ideal final code transition of the A/D converter. R_{GTRIM} is then adjusted until 50 percent of the conversion results are 3FEH and 50 percent are 3FFH. Once this adjustment is complete, the converter zero offset and full-scale errors are nulled, and could be ignored (except for temperature variation). This allows the system to rely upon the tighter, more descriptive converter specifications for Terminal Based Non-Linearity and Differential Non-Linearity.

The circuits of Figure 13 show only feedback resistors that set the gain of the buffer. In practice, it will often be necessary to include offset adjustment, gain trimming, temperature or frequency stability compensation, or components to build an active filter.

Figure 14 depicts a generalized non-inverting input buffer that offsets the analog input and scales the input

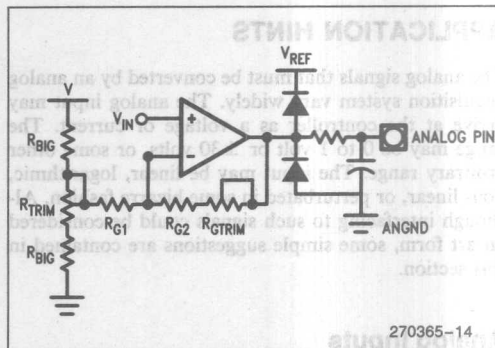


Figure 14. Trimming Offset and Gain

Analog References

Reference supply levels strongly influence the absolute accuracy of the conversion. For this reason, it is recommended that the ANGND pin be tied to a clean ground, as close to the power supply as possible. Bypass capacitors should also be used between V_{REF} and ANGND. ANGND should be connected to V_{SS} only at the chip. V_{REF} should be well regulated and used only for the A/D converter. The V_{REF} supply can be between 4.5V and 5.5V and needs to be able to source around 5 mA. Figure 15 shows all of these connections.

Note that if only ratiometric information is desired, V_{REF} can be connected to V_{CC} . In addition, if the A/D converter is not being used, V_{REF} must be connected to V_{CC} and ANGND to V_{SS} for Port0 to work as a digital port.

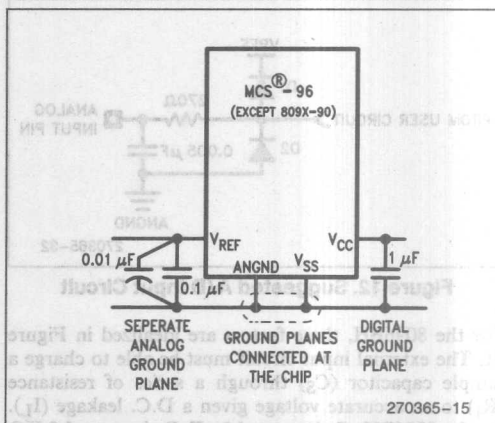


Figure 15. Supply Decoupling

Getting More Resolution

Given that the A/D converter can convert an analog input ranging from 0 volts to 5 volts into 1024 steps of 5 millivolts each, the desire for more resolution can come from three basic needs – need extra LSB, need extra MSB, need BOTH.

The configuration shown in Figure 16 can be used to solve each of the “more resolution” problems. This set-up requires the use of two input channels with different offsets and gains.

When the 5 millivolt step size of the A/D is too large for the application requirements, but the 5 volt range is sufficient, the system needs an “extra LSB”. For example, an application requiring 2.5 millivolt steps over a 5 volt range needs an 11-bit conversion result. The 11th bit needs to be added to the least significant side of the 10-bit result (the “right”). This can be achieved using the circuit of Figure 16.

If both channels are set for a gain of 2, with channel 1 offset to 2.5 volts, the 5 volt input range is split into 2.5 volt ranges that are amplified by two before being input to the A/D. While V_{IN} is between 0 and 2.5 volts, channel 0 will be providing a proportional voltage between 0 volts and 5 volts to the A/D converter. Channel 1 will be clamped to 5 volts. When V_{IN} rises above 2.5 volts, channel 1 will begin to output a proportional voltage between 0 volts and 5 volts to the A/D converter and channel 0 will be clamped at 5 volts. Using this method, an 11-bit (2048 step) result is created with 2.5 millivolt steps (i.e. an extra LSB).

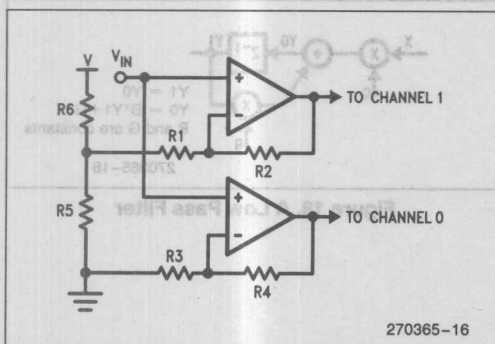


Figure 16. A Flexible Input Circuit

It is useful to note that only one conversion per sample will be required if the software keeps track of which channel is active. The only time that two conversions will be required for one sample is when the voltage crosses the midpoint.

The second reason that “more resolution” is requested is the need for an “extra MSB”. When the converter’s input voltage range is too small (5 volts when 10 volts is needed), but 5 millivolt steps over the actual input voltage range is sufficient, an extra bit is needed on the most significant (“left”) side of the 10-bit result. The circuit of Figure 16 can also be used, with different gains and offsets, to satisfy this extra MSB need by splitting the 10 volt range into 5 volt ranges.

If both channels of Figure 16 are set for unity gain, and channel 1 is offset to 5 volts, an 11-bit conversion result with 5 millivolt steps is available. While V_{IN} is in the lower half of its range (0 volts to 5 volts), channel 0 will be active. While V_{IN} is in the upper half of its range (5 volts to 10 volts), channel 1 will be active. Thus, an extra MSB is created. For applications requiring multiple extra bits of result, the solutions can become more “elegant” (i.e. elaborate). However, it is profitable to first squeeze the most out of the now familiar circuit in Figure 16.

Assume that the analog input, V_{IN} , ranges from 0 volts to 10 volts, and it is desired to measure this range in 2.5 millivolt steps. This requires two extra bits of result – one extra MSB and one extra LSB. A simple extrapolation of the preceding discussion of creating extra bits might have the designer planning to tie up four channels of the multiplexer needlessly. Needlessly, that is, if the application is a typical control application where the high accuracy requirements are only important in the “normal” operating range of the process. Outside of the normal operating range is the “possible” operating range which must be measured, but with less stringent requirements.

Since the requirements of the normal range set the necessary LSB weight, and the extent of the possible range sets the maximum voltage span, it follows that only two channels need to be used (Figure 16). Channel 0 would be set with a gain that compressed the possible V_{IN} range to 5 volts, while channel 1 would be offset to the normal operating range and would have a gain of two to expand this region of critical interest. With this ap-

proach, 100 percent of the normal operating range is digitized in 2.5 millivolt steps, while 100 percent of the possible range is digitized in 10 millivolt steps.

Unfortunately, not all high resolution applications can be described as a process with a small region of in-control operation, where the process is out-of-control outside of that small region. For example, it is necessary to measure airflow in an engine controlling carburetion. The air flow at idle is likely to be several orders-of-magnitude lower than the airflow at full RPM. The process needs to be in tight control over the entire range, not only when the engine is at half-speed.

When it is desired to measure a process with a fixed percent of error throughout a range spanning several orders-of-magnitude, a non-linear input buffer becomes attractive. For example, assume that the analog signal that needs to be digitized can vary from 1 millivolt to 25 volts and describes a physical process that must be represented digitally with 1 percent error at any point in the possible input range. A linear solution to this application would require a converter with a 10 microvolt LSB ($1\% \times 1 \text{ mV}$), and a resolution of 22 bits ($25 \text{ V}/10 \text{ microvolts}$). This is clearly undesirable.

The use of a log input buffer to compress the 25 volt range logarithmically to 5 volts would satisfy the application requirements. The input would range from 1 millivolt to 25 volts with the output ranging from 0 volts to 5 volts proportionally to the log of $V_{IN}/1\text{mV}$. Each one-percent change in the input voltage would change the output voltage by 5 millivolts (one count). The antilog could be taken in software using a lookup table, or the control calculations could be performed in a log base.

Simple inexpensive log-amps can be built as in Figure 17, or high-accuracy, self-contained log-amps can be purchased. Which is chosen depends upon the application tradeoffs of price and performance.

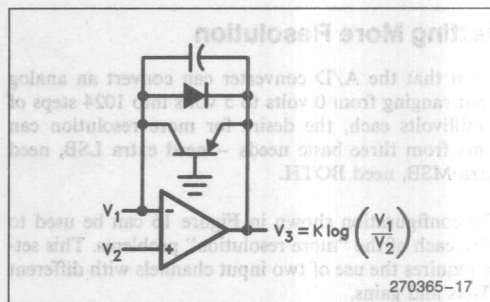


Figure 17. A Low-Cost Log Amplifier

Other techniques become available for consideration in systems that have slow sample rate requirements, but very high resolution requirements. In addition to the methods described above, which require external hardware, software filtering or other post-processing of the conversion results can be productive. Each method relies upon the ability to sample the analog input much faster than the system requires an analog input.

When resolution is limited by filterable noise, perhaps the most straightforward approach to post-processing is to oversample the input by a factor of N and digitally low-pass filter the data (i.e. weighted rolling average). A result would be reported to the rest of the system every N samples (Figure 18). A low-pass filter can increase the signal-to-noise ratio (SNR) by a factor of N (see bibliography). However, care must be taken to be certain that the input voltage varies slowly with respect to the sampling rate.

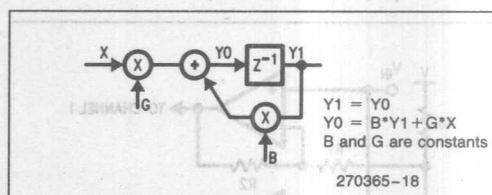


Figure 18. A Low Pass Filter

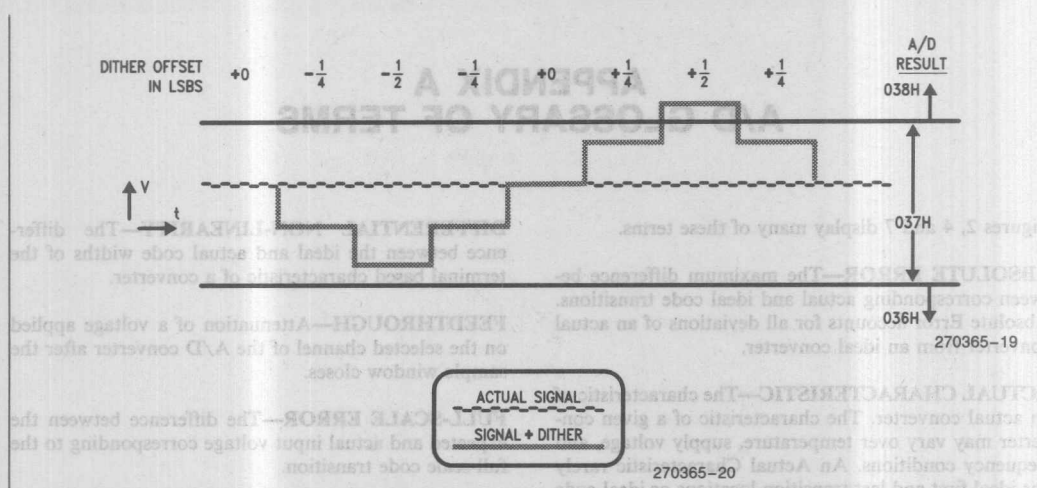


Figure 19. Dither

Another approach to creating more resolution is called "synchronized dither". Figure 19 shows an input voltage that is constant somewhere between two code transition points. This input is "dithered" by adding a small periodic waveform ($1/4$ LSB steps) to the input while performing an A/D conversion synchronized with each dither step. Every time the dither completes a full cycle, the eight conversion results are averaged to form one digitized value. Since the dither is periodic and symmetrical about 0 volts, its average impact on the input voltage is 0 volts.

The creation of extra resolution can be seen with the example shown in Figure 19. Without dither, the input voltage would always convert to 37H. With dither, one-eighth of the conversions would be 38H and $7/8$ of the conversions would be 37H. If every eight conversions were averaged, the result would be $37H + 1/8$ LSB. The possible results given a four level dither, where the input voltage was always within the 37H code width, would be

$$\begin{aligned} &36H + 5/8 \\ &36H + 7/8 \\ &37H + 0 \\ &37H + 1/8 \\ &37H + 3/8 \end{aligned}$$

Hence, four new levels exist (two bits).

Dither will only create more resolution up to the limit of the A/D converter comparator's ability to distinguish voltages. Since MCS-96 converter repeatability error is typically around 1 millivolt to 1.25 millivolts, $1/4$ LSB dither is the practical limit if no other processing is done. Figure 20 shows a simple method by which

the input voltage could be dithered under software control.

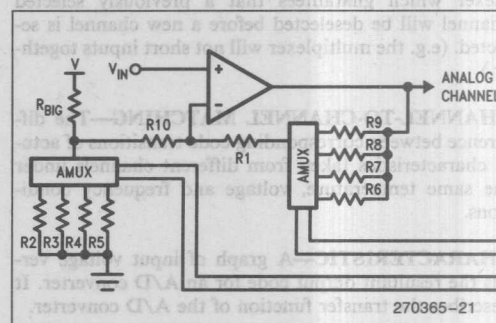


Figure 20. Software Controlled Offset and Gain

While only a few of the more obvious interfacing techniques were described here, there are as many innovative interfacing tricks as there are designers.

CONCLUSION

This application note provides a fundamental understanding of MCS-96 analog acquisition for the digital designer. Since answering the limitless number of analog circuit design questions is beyond the scope of this document, it is expected that analog design manuals and the large body of publicly available applications literature will be consulted for detailed design hints. Furthermore, the applications literature of monolithic analog acquisition system manufacturers should be consulted since the suggestions presented therein are largely transportable to any A/D system.

APPENDIX A A/D GLOSSARY OF TERMS

Figures 2, 4 and 7 display many of these terms.

ABSOLUTE ERROR—The maximum difference between corresponding actual and ideal code transitions. Absolute Error accounts for all deviations of an actual converter from an ideal converter.

ACTUAL CHARACTERISTIC—The characteristic of an actual converter. The characteristic of a given converter may vary over temperature, supply voltage, and frequency conditions. An Actual Characteristic rarely has ideal first and last transition locations or ideal code widths. It may even vary over multiple conversions under the same conditions.

BREAK-BEFORE-MAKE—The property of a multiplexer which guarantees that a previously selected channel will be deselected before a new channel is selected. (e.g. the multiplexer will not short inputs together.)

CHANNEL-TO-CHANNEL MATCHING—The difference between corresponding code transitions of actual characteristics taken from different channels under the same temperature, voltage and frequency conditions.

CHARACTERISTIC—A graph of input voltage versus the resultant output code for an A/D converter. It describes the transfer function of the A/D converter.

CODE—The digital value output by the converter.

CODE CENTER—The voltage corresponding to the midpoint between two adjacent code transitions.

CODE TRANSITION—The point at which the converter changes from an output code of Q , to a code of $Q + 1$. The input voltage corresponding to a code transition is defined to be that voltage which is equally likely to produce either of two adjacent codes.

CODE WIDTH—The voltage corresponding to the difference between two adjacent code transitions.

CROSSTALK—See "Off-Isolation".

D.C. INPUT LEAKAGE—D.C. Leakage current of an analog input pin.

DIFFERENTIAL NON-LINEARITY—The difference between the ideal and actual code widths of the terminal based characteristic of a converter.

FEEDTHROUGH—Attenuation of a voltage applied on the selected channel of the A/D converter after the sample window closes.

FULL-SCALE ERROR—The difference between the expected and actual input voltage corresponding to the full-scale code transition.

IDEAL CHARACTERISTIC—A characteristic with its first code transition at $V_{IN} = 0.5 \text{ LSB}$, its last code transition at $V_{IN} = (V_{REF} - 1.5 \text{ LSB})$ and all code widths equal to one LSB.

INPUT RESISTANCE—The effective series resistance from the analog input pin to the sample capacitor.

LSB - LEAST SIGNIFICANT BIT—The voltage value corresponding to the full-scale voltage divided by 2^n , where n is the number of bits of resolution of the converter. For a 10-bit converter with a reference voltage of 5.12 volts, one LSB is 5.0 mV. Note that this is different than digital LSBs, since an uncertainty of two LSBs, when referring to an A/D converter, equals 10 mV. (This has been confused with an uncertainty of two digital bits, which would mean four counts, or 20 mV.)

MONOTONIC—The property of successive approximation converters which guarantees that increasing input voltages produce adjacent codes of increasing value, and that decreasing input voltages produce adjacent codes of decreasing value.

NO MISSED CODES—For each and every output code, there exists a unique input voltage range which produces that code only.

NON-LINEARITY—The maximum deviation of code transitions of the terminal based characteristic from the corresponding code transitions of the actual characteristic of a converter.

OFF-ISOLATION—Attenuation of a voltage applied on a deselected channel of the A/D converter. (Also referred to as Crosstalk.)

REPEATABILITY—The difference between corresponding code transitions from different actual characteristics taken from the same converter on the same channel at the same temperature, voltage and frequency conditions.

RESOLUTION—The number of input voltage levels that the converter can unambiguously distinguish between. Also defines the number of useful bits of information which the converter can return.

SAMPLE DELAY—The delay from receiving the start conversion signal to when the sample window opens.

SAMPLE DELAY UNCERTAINTY—The variation in the Sample Delay.

SAMPLE TIME—The time that the sample window is open.

SAMPLE TIME UNCERTAINTY—The variation in the sample time.

During the sample window (Figure B1a), V_{ANIN} and V_{OPG} control the amount of charge stored in C_A and C_B (V_{OPG} controls the converter offset). Once the sample window closes (Figure B1b), voltages applied to V_{IN} and V_{IN2} will add or subtract charge proportional to $(V_{ANIN} - V_{IN})$ on C_A and $(V_{OPG} - V_{IN2})$ on C_B . Unless a voltage is applied to V_{IN} and V_{IN2} , the inverting comparator input of Figure B1b will remain at V_{BIAS} due to the charges on C_A and C_B . The non-inverting comparator input will always remain at V_{BIAS} and serves as a reference.

If a V_{IN} combination is applied which causes the non-inverting input to drop below V_{BIAS} , the comparator will output a 1 to indicate that the applied voltage was lower than the original V_{ANIN} . To better understand how the circuit works, Figure B2 shows the superposition analysis used to form the equation for V_{OUT} , given initial charge on C_A and C_B and new input voltages V_{IN} and V_{IN2} .

SAMPLE WINDOW—Begins when the sample capacitor is attached to a selected channel and ends when the sample capacitor is disconnected from the selected channel.

SUCCESSIVE APPROXIMATION—An A/D conversion method which uses a binary search to arrive at the best digital representation of an analog input.

TEMPERATURE COEFFICIENTS—Change in the stated variable per degree centigrade temperature change. Temperature coefficients are added to the typical values of a specification to see the effect of temperature drift.

TERMINAL BASED CHARACTERISTIC—An Actual Characteristic which has been rotated and translated to remove zero offset and full-scale error.

VCC REJECTION—Attenuation of noise on the V_{CC} line to the A/D converter.

ZERO OFFSET—The difference between the expected and actual input voltage corresponding to the first code transition.

Before beginning a detailed description of the capacitive part of the conversion process, it is necessary to understand a few details about the resistor chain.

There are 256 resistors connected in series from the analog reference to analog ground. The actual value of the resistors only impacts the current through the resistor chain. If every resistor in the chain is the same value, the converter will function properly.

To reduce resistor-to-resistor variation, the chain is folded in half and then in an accordion fashion to produce a 16×16 block of resistors. This minimizes the sensitivity of the array to processing gradients, while also allowing the array to be addressed roughly similar to a 16×16 memory array.

APPENDIX B CAPACITIVE INTERPOLATION

A successive approximation A/D converter needs an internal D/A converter of the same resolution as the desired A/D result. A 10-bit D/A could have been made using a string of 1024 resistors connected from the analog reference at one end to ground at the other end. Although this would be technically ideal, such a circuit would be enormous. Therefore, a method was developed to generate the needed reference voltages using a small area of silicon so that an on-chip 10-bit A/D converter would be economical.

The method used relies upon a 256-resistor chain to generate reference voltages in 20mV ($5.12\text{V}/256$) steps while two ratioed capacitors are used to capacitively "interpolate" voltages in-between the resistor tap voltages. The area of the 256-resistor chain together with the capacitors is one-fourth the area of the would-be 1024 resistor chain.

Before beginning a detailed description of the capacitive part of the conversion process, it is necessary to understand a few details about the resistor chain.

There are 256 resistors connected in series from the analog reference to analog ground. The actual value of the resistors only impacts the current through the reference pin. If every resistor in the chain is the same value the converter will function properly.

To reduce resistor-to-resistor variation, the chain is folded in half, and then in an accordion fashion to produce a 16×16 block of resistors. This minimizes the sensitivity of the array to processing gradients, while also allowing the array to be addressed roughly similar to a 16×16 memory array.

REPEATABILITY—The difference between corresponding code transitions from different actual charge levels taken from the same converter on the same silicon at the same temperature, voltage and humidity conditions.

RESOLUTION—The number of input voltage levels that the converter can unambiguously distinguish between. Also defines the number of useful bits of information.

As explained earlier, it is desired for the A/D converter to have its first code transition at $\frac{1}{2}$ LSB followed by subsequent code widths 1 LSB wide.

To accomplish this, each resistor is tapped in its center rather than between resistors. For example, the first resistor tap is half-way up the first resistor. This means that the zero resistor tap will output 10mV ($20\text{mV}/2$). When calculating the voltage on a certain resistor tap, you must add 10mV to the product of the tap number and 20mV.

The internal connections while an analog input is being sampled are shown in Figure B1a. Once sampling is complete, the analog input is disconnected and the comparator inputs are no longer clamped to V_{BIAS} (Figure B1b).

During the sample window (Figure B1a), V_{ANIN} and V_{OFS} control the amount of charge stored in C_A and C_B (V_{OFS} controls the converter offset). Once the sample window closes (Figure B1b), voltages applied to V_{IN} and V_{IN2} will add or subtract charge proportional to $(V_{\text{ANIN}} - V_{\text{IN}})$ on C_A and $(V_{\text{OFS}} - V_{\text{IN2}})$ on C_B . Unless a voltage is applied to V_{IN} and V_{IN2} , the inverting comparator input of Figure B1b will remain at V_{BIAS} due to the charges on C_A and C_B . The non-inverting comparator input will always remain at V_{BIAS} and serves as a reference.

If a V_{IN} , V_{IN2} combination is applied which causes the non-inverting input to drop below V_{BIAS} the comparator will output to a 1 to indicate that the applied voltage was lower than the original V_{ANIN} . To better understand how the circuit works, Figure B2 shows the superposition analysis used to form the equation for V_{OUT} , given initial charge on C_A and C_B and new input voltages V_{IN} and V_{IN2} .

Adding the independent effects shown in Figure B2 we have:

$$V_{OUT} = V_1 + V_2 + V_3 + V_4$$

$$V_{OUT} = V_{IN} \left(\frac{C_A}{C_A + C_B} \right) + V_{IN2} \left(\frac{C_B}{C_A + C_B} \right) + V_{AI} \left(\frac{C_A}{C_A + C_B} \right) + V_{BI} \left(\frac{C_B}{C_A + C_B} \right)$$

$$V_{OUT} = (V_{IN} + V_{AI}) \frac{C_A}{C_A + C_B} + (V_{IN2} + V_{BI}) \frac{C_B}{C_A + C_B}$$

The initial conditions on C_A and C_B are set-up as shown in Figure B3.

We can see that:

$$V_{AI} = V_{BIAS} - V_{ANIN} \quad (II)$$

$$V_{BI} = V_{BIAS} - V_{OFS} \quad (III)$$

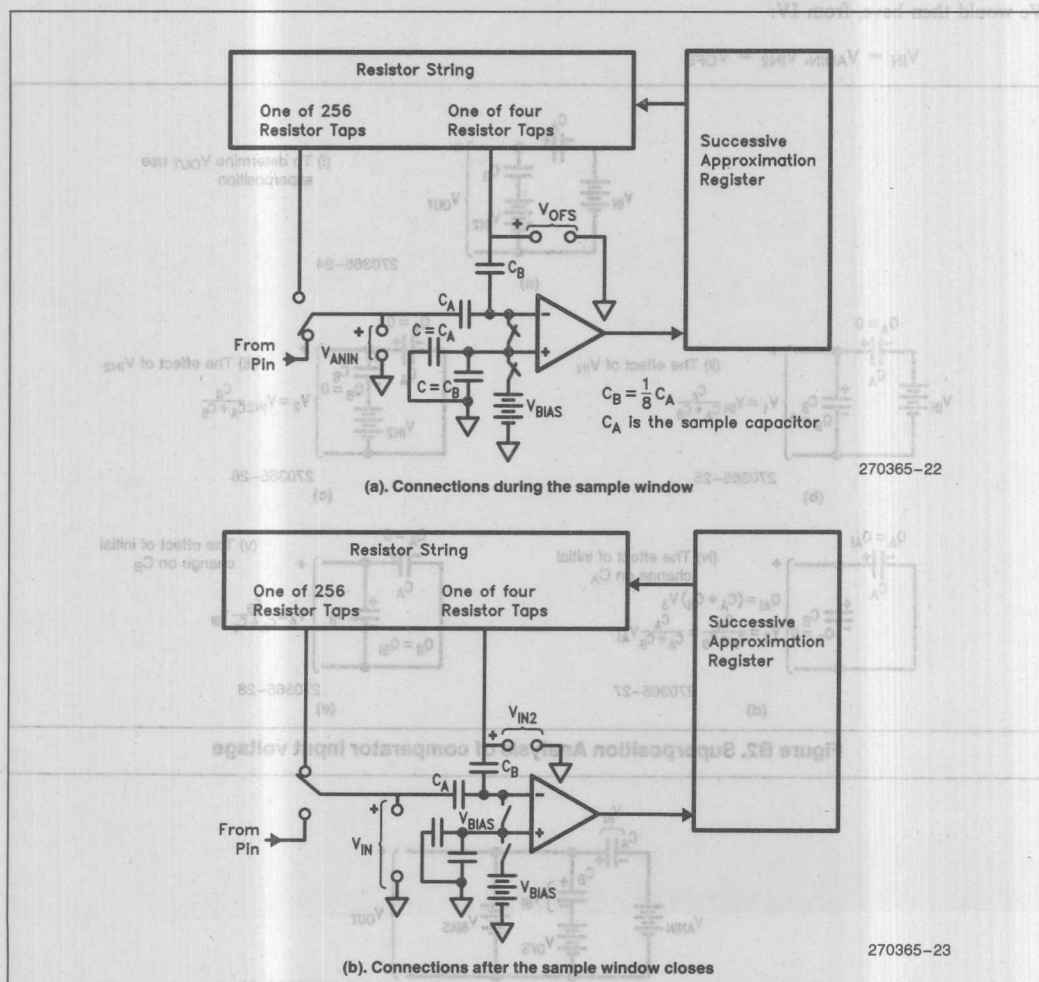


Figure B1

Substituting II and III into I we get:

$$V_{OUT} = (V_{IN} + V_{BIAS} - V_{ANIN}) \frac{C_A}{C_A + C_B} + (V_{IN2} + V_{BIAS} - V_{OFS}) \frac{C_B}{C_A + C_B} \quad (IV)$$

V_{OUT} becomes the input voltage to the comparator which ideally presents no load. The only way to make V_{OUT} approach the value of V_{BIAS} (after V_{BIAS} is removed) is to apply a voltage combination which makes equation IV evaluate to V_{BIAS} . If we had an infinitely variable internal voltage reference to use, we could just set the reference on V_{IN} to the value of V_{ANIN} and make $V_{IN2} = V_{OFS}$.

We would then have, from IV:

$$V_{IN} = V_{ANIN}, V_{IN2} = V_{OFS}$$

However, using a 256-resistor chain to provide references, we can find a V_{IN} , V_{IN2} combination which can bring V_{OUT} close to the value of V_{BIAS} . The 256-resistor chain provides a reference voltage in 20 mV steps. We can then take separate taps of the resistor chain and connect them to V_{IN} and V_{IN2} . The voltage attached to V_{IN} will couple to V_{OUT} by a factor of $C_A/(C_A + C_B) = 8/9$ from EQN IV. The voltage attached to V_{IN2} will couple to V_{OUT} by a factor of $C_B/(C_A + C_B)$. The ratio of the impacts on V_{OUT} of V_{IN} versus V_{IN2} is:

$$\left(\frac{\partial V_{OUT}}{\partial V_{IN}} \right) \div \left(\frac{\partial V_{OUT}}{\partial V_{IN2}} \right) = (8/9)/(1/9) = 8$$

Therefore, a voltage change on V_{IN} will affect the voltage seen at V_{OUT} eight times more than the same change placed on V_{IN2} .

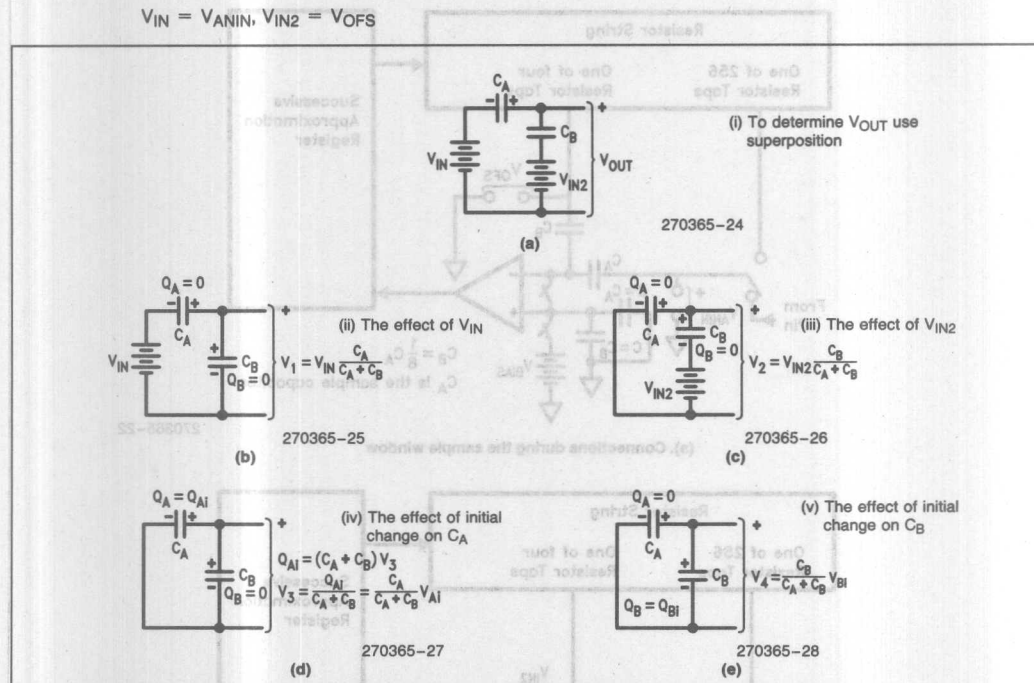


Figure B2. Superposition Analysis of comparator input voltage

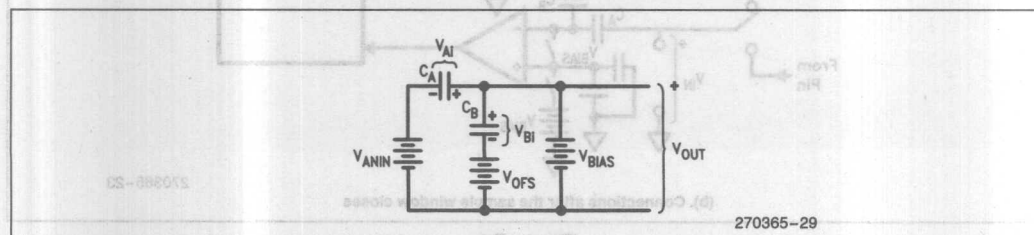


Figure B3. Initial Conditions

For example, assume the actual input voltage V_{ANIN} was 2.50mV during the sample window. Using EQN IV, and assuming $V_{BIAS} = 3V$ and $V_{OFS} = 70mV$, we substitute and find:

$$V_{OUT} = (V_{IN} + 2.9975) \times (8/9) + (V_{IN2} + 2.93) \times (1/9) \quad (V)$$

Using successive approximation, the first trial input voltage attempted corresponds to the digital code 0111 1111 11b (127 \times 20mV + 10mV). This means that the voltage applied to V_{IN} will be the 0111 1111b tap and the voltage applied to V_{IN2} will be the 0110b tap (6 \times 20mV + 10mV = 3 LSB). Substituting these values into EQN V we have:

$$V_{OUT} = (2.550 + 2.9975) \times (8/9) + (0.130 + 2.93) \times (1/9) \quad (V)$$

$$V_{OUT} = 4.931 + 0.34 = 5.271 \quad (V)$$

Since the 3V reference is lower than V_{OUT} with these inputs, the comparator will output a 0 which is placed in the MSB of the successive approximation register. The next most significant bit of the SAR is then zero'd

and the new ladder tap applied to V_{IN} . The result of this second comparison, and the subsequent comparisons are shown in Table B1. The C program used to generate Table B1 is listed in Listing B1.

The value selected for V_{OFS} during the sample window may not be obvious. The purpose of V_{OFS} is to inject a constant offset in the sampling process so that the converter's first code transition will occur at 2.5mV.

Using EQN IV we can quickly see why V_{OFS} is chosen to be the fourth resistor tap (4 \times 20mV + 10mV = 70mV). For $V_{ANIN} = 2.5mV$, we want V_{OUT} to evaluate to V_{BIAS} when the SAR is OH.

$$V_{OUT} = \{(0.20 \text{ mV} + 10 \text{ mV}) + (V_{BIAS} - 2.5 \text{ mV})\} \times (8/9) + \{(0.20 \text{ mV} + 10 \text{ mV}) + (V_{BIAS} - 70 \text{ mV})\} \times (1/9)$$

$$V_{OUT} = V_{BIAS} = 7.5 \text{ mV} \times (8/9) - 60 \text{ mV} \times (1/9) = 0$$

Therefore, if $V_{OFS} = 70 \text{ mV}$, the converter's first code transition will be when $V_{ANIN} = 2.5 \text{ mV}$.

Table B1. Conversion Simulation

A to D simulator. (center taps) .. With	
$V_{IN} = 0.002500$	
$V_{CENT} = 3.000000$ $V_{OFF} = 0.070000$	
SAR = 1FFH (511)	$V_{OUT} = 5.271111$
SAR = FFH (255)	$V_{OUT} = 4.133333$
SAR = 7FH (127)	$V_{OUT} = 3.564444$
SAR = 3FH (63)	$V_{OUT} = 3.280000$
SAR = 1FH (31)	$V_{OUT} = 3.137778$
SAR = FH (15)	$V_{OUT} = 3.066667$
SAR = 7H (7)	$V_{OUT} = 3.031111$
SAR = 3H (3)	$V_{OUT} = 3.013333$
SAR = 1H (1)	$V_{OUT} = 3.004444$
SAR = OH (0)	$V_{OUT} = 3.000000$
SAR = 1H (1)	which means 0.005000 volts

```

#include "CTYPE.H"
#include "STDIO.H"
/* example invocation lines

a2dsim 0.0025 3.0 0.07 p
Vin Vbias Vofs print to screen and lp

a2dsim 0.0075 3.0 0.07
Vin Vbias Vofs print to screen only

*/
int main(k, argv)
int k;
char *argv[];
{
    FILE *fp, *fopen();
    double initial_conditions, vin, vout, vcent, voff, v89, v19;
    unsigned int sar = 0x3FF;
    unsigned int mask = 0x200;
    unsigned int count = 0;
    unsigned int printon;
    if (strcmp(argv[0], "run") == 0)
        count++;
    if ((k != (4 + count)) & (k != (5 + count)))
    {
        printf("\nInvocation error!\n");
        return;
    }
    count++;
    sscanf(argv[count++], "%lf", &vin);
    sscanf(argv[count++], "%lf", &vcent);
    sscanf(argv[count++], "%lf", &voff);
    if (count == k)
        printon = 0;
    else printon = 1;
    printf("A to D simulator.(center taps..");
    if (printon)
    {
        if ((fp = fopen("prn:", "w")) == 0)
        {
            printf("\nCan't open printer\n");
            return;
        }
    }
    if (printon)
        fprintf(fp, "A to D simulator..");

    printf(" with \nVin = %f\nVcent = %f\nVoff = %f\n", vin, vcent, voff);
    if (printon)
        fprintf(fp, " with \nVin = %f\nVcent = %f\nVoff = %f\n",
            vin, vcent, voff);

    initial_conditions = ((8.0 / 9.0) * (vcent - vin))
        + ((1.0 / 9.0) * (vcent - voff));
    v89 = 8.0 / 9.0;
    v19 = 1.0 / 9.0;

```

270365-A5

Listing B1. A/D Converter Simulator

```

sar ^= mask;
printf("SAR = %3xH (%4d)\t", sar, sar);
if (prnton)
    fprintf(fp, "SAR = %3xH (%4d)\t", sar, sar);
for (count = 0; count < 10; count++)
{
    vout = (v09 * (((double) (sar >> 2)) * 0.02 + 0.01))
        + (v19 * (((double) ((sar & 3) << 1)) * 0.02 + 0.01))
        + initial_conditions;
    if (vout < vcent)
        sar |= mask;
    mask >>= 1;
    sar ^= mask;
    printf("Vout = %f\nSAR = %3xH (%4d)\t", vout, sar, sar);
    if (prnton)
        fprintf(fp, "Vout = %f\nSAR = %3xH (%4d)\t",
            vout, sar, sar);
}
printf(" which means %f volts\n\n", (double) sar * 0.005);
if (prnton)
    fprintf(fp, " which means %f volts\n\n", (double) sar * 0.005);
return;
}
/* main */

```

270365-A6

Listing B1. A/D Converter Simulator (Continued)

APPENDIX C ERROR FORMULAS

The following C program listing contains the routines used to calculate A/D performance in the Embedded Controller Applications lab. Most of the routines require floating point arrays to operate upon. In the listings, the array `x[]` contains the input voltages corresponding to each code transition of the converter. The array `dx[]` contains the width of the region in which each code transition of the converter could occur. For example, an input voltage of 0.003V may cause code 0 and code 1 to be equally likely outputs. `x[0]` would then contain 0.0030000. However, 0-to-1 code transitions might be observed infrequently through a range of input voltages from 0.0025V to 0.0035V. `dx[0]` would then contain 0.0010000 to indicate that there is a 1 millivolt window in which either code could occur. `x[]` and `dx[]` are generated by hardware doing repeat-

ed conversions using precision voltage standards to provide the input voltages. The array `dd[]` is used throughout as temporary storage.

Generally, typical data is drawn from `x[]` only. When minimum and maximum data is desired, `x[]` and `dx[]` are used to find the range of possible input voltages that could cause each code. For example, typical zero offset is found by simply subtracting 0.5 LSB from the value of `x[0]`. But, the minimum and maximum zero offset would be calculated as $x[0] - 0.5 \text{ LSB} \pm dx[0]/2$.

The listings are provided to show exactly how performance data is calculated. They are not meant to be compiled by the reader. In fact, they are too incomplete to compile correctly, as some support routines and global data structures are not provided.


```

#include "DPR\ADTMAC.H"
#include "DPR\YDBASE.H"
#include "DPR\RDBASE.H"
#define LSB (now.avcc/(pow(2,nbits)))
#define FCT (int)(pow(2,nbits) - 2)
#undef min
#undef max
#undef abs

double pow(a, b)
int a, b;
{
    double temp;
    int i;
    temp = 1.0;
    for (i = 1; i <= ((int) b); i++, temp = temp * a)
        ;
    return (temp);
}

double fabs(a)
double a;
{
    if (a < 0)
        return (-a);
    else return (a);
}

int min(a, b)
double a, b;
{
    if (a < b)
        return (1);
    else if (a > b)
        return (2);
    else return (0);
}

int max(a, b)
double a, b;
{
    return (min(b, a));
}

double typgoff(x, dx)
float x[], dx[];
{
    double povf();
    return (x[0] - (0.5 * LSB));
}

double maxgoff(x, dx)
float x[], dx[];
{
    double povf();
    return (x[0] + (dx[0] / 2.0) - 0.5 * LSB);
}

double mingoff(x, dx)

```

270365-A7

8A-282013

Listing C1. Error Formulas

Listing C1. Error Formulas (Continued)

```

float x[], dx[];
{
    double pow();
    return (x[0] - (dx[0] / 2.0) - 0.5 * LSB);
}

double typfse(x, dx)
float x[], dx[];
{
    double pow();
    return (x[FCT] - (nov.avcc - (1.5 * LSB)));
}

double minfse(x, dx)
float x[], dx[];
{
    double pow();
    return ((x[FCT] - (dx[FCT] / 2.0)) - (nov.avcc - (1.5 * LSB)));
}

double maxfse(x, dx)
float x[], dx[];
{
    double pow();
    return ((x[FCT] + (dx[FCT] / 2.0)) - (nov.avcc - (1.5 * LSB)));
}

int kaberror(x, dx, dd, start, stop) /* transition absolute error */
float x[], dx[], dd[];
unsigned int start, stop;
{
    double pow(), fabs();
    int i, worst;
    for (i = worst = start; i <= stop; i++)
    {
        dd[i] = x[i] - ((double) i + 0.5) * LSB;
        if (fabs(dd[i]) > fabs(dd[worst]))
            worst = i;
    }
    return (worst);
}

int kaberrordx(x, dx, dd, start, stop) /* transition absolute error w/dx */
float x[], dx[], dd[];
unsigned int start, stop;
{
    double pow(), fabs();
    int i, worst;
    double t1, t2;
    for (i = worst = start; i <= stop; i++)
    {
        t1 = (x[i] - (dx[i] / 2.0)) - (((double) i + 0.5) * LSB);
        t2 = (x[i] + (dx[i] / 2.0)) - (((double) i + 0.5) * LSB);
        if (fabs(t1) > fabs(t2))
            dd[i] = t1;
        else dd[i] = t2;
        if (fabs(dd[i]) > fabs(dd[worst]))
            worst = i;
    }
    return (worst);
}

```

Listing C1. Error Formulas (Continued)


```

for (i = worst = start; i <= stop; i++)
{
    t1 = (x[i] - (dx[i] / 2.0) - oadj) * qadj - (((double) 1 + 0.5) * LSB);
    t2 = (x[i] + (dx[i] / 2.0) - oadj) * qadj - (((double) 1 + 0.5) * LSB);
    if (fabs(t1) > fabs(t2))
        dd[i] = t1;
    else dd[i] = t2;
    if (fabs(dd[i]) > fabs(dd[worst]))
        worst = i;
}
return (worst);
}

int rdnl(x, dx, dd, start, stop) /* using x only */
float x[], dx[], dd[];
int start, stop;
{
    int i, worst;
    double pow(), fabs();
    double oadj, qadj;
    double typfse(), typzoff();

    oadj = typzoff(x, dx);
    qadj = 1.0 + ((typfse(x, dx) - oadj) / x[stop]);

    worst = start;
    if (start == 0)
    {
        dd[0] = 0.0;
        start++;
    }
    for (i = start; i <= stop; i++)
    {
        dd[i] = (x[i] - oadj) * qadj
            - (x[i - 1] - oadj) * qadj
            - LSB;

        if (fabs(dd[i]) > fabs(dd[worst]))
            worst = i;
    }
    return (worst);
}

int rdnlx(x, dx, dd, start, stop) /* using x and dx */
float x[], dx[], dd[];
int start, stop;
{
    int i, worst;
    double pow(), fabs();
    double t1, t2;
    double oadj, qadj;
    double typfse(), typzoff();

    oadj = typzoff(x, dx);
    qadj = 1.0 + ((typfse(x, dx) - oadj) / x[stop]);

    worst = start;
    if (start == 0)
    {
        dd[0] = dx[0] / 2.0;

```

270365-B0

Listing C1. Error Formulas (Continued)


```

start++;
for (i = start; i <= stop; i++)
{
    t1 = (x[i] - (dx[i] / 2.0) - oad) * gad;
    - (x[i - 1] + (dx[i - 1] / 2.0) - oad) * gad;
    - LSB;
    t2 = (x[i] + (dx[i] / 2.0) - oad) * gad;
    - (x[i - 1] - (dx[i - 1] / 2.0) - oad) * gad;
    - LSB;
    if (fabs(t1) > fabs(t2))
        dd[i] = t1;
    else dd[i] = t2;
    if (fabs(dd[i]) > fabs(dd[vorst]))
        vorst = i;
}
return (vorst);
}

```

```

int reslevels(x, dx) /* finds resolution in levels */
float x[], dx[]:
{
    int i, levels, n;
    double pow();

    levels = 1;
    n = (int) pow(2, nbits) - 1;
    if ((x[0] - (dx[0] / 2.0) > 0.0))
        levels++;
    for (i = 1; i < n; i++)
        if ((x[i] - 1) + (dx[i] / 2.0) > 0.0)
            levels++;
    return (levels);
}

```

270365-B1

Listing C1: Error Formulas (Continued)

Resolution is 1024 levels.

Maximum Reliability Error 301 = 0.001875
 Minimum Reliability Error 301 = 0.000934

Term. Non. Lin. Error (min) 40 = 0.002148
 Term. Non. Lin. Error (max) 302 = -0.004103

Diff. Non. Lin. Error (min) 75 = -0.001071
 Diff. Non. Lin. Error (max) 40 = 0.003747

Absolute Error (min) 302 = 0.00111
 Absolute Error (max) 40 = 0.004785
 Absolute Error (typ) 40 = 0.004137

Minimum FS Error = -0.000120
 Maximum FS Error = -0.001024
 Typical FS Error = -0.000388

Maximum Offset Error = -0.001385
 Maximum Offset Error = -0.001023

APPENDIX D SAMPLE CONVERTER DATA

The following pages include printouts describing the performance of an 8097BH. The data shown is for one device and is provided for illustrative purposes only. Users should only rely upon data sheet specifications for the exact device they are designing with.

Table D1 summarizes many performance measures for one converter at 25 C, 12 MHz, $V_{CC} = 5.00$ volts and

$V_{REF} = 5.120$ volts. Following Table D2 are several error plots that describe Absolute Error, Terminal-based Non-Linearity, Differential Non-Linearity and Repeatability for the test device code-by-code. The y-axis in the plots is the error in volts for each code transition, where code transitions make up the x-axis.

Table D1. Sample Converter Data

```

Test ID = DOH
sN: 4130 (1022H)
T = 25.000000
VCC = 5.000000, AVCC = 5.120000
Freq = 12.000000
Chan. = 3
States = 188 Mode = 0H
X0.15 1/28/87
Transition Characterization Parameter Listing
Large Step = 0.001000 V
Small Step = 0.000100 V
Endpoints when (1/100) are wrong

Center is 50 percent

Typical Offset Error = -0.001923
Maximum Offset Error = -0.002460
Maximum Offset Error = -0.001385

Typical FS Error = -0.000566
Maximum FS Error = -0.001254
Minimum FS Error = -0.000120

Absolute Error (typ) 40 = 0.004157
Absolute Error (max) 40 = 0.004795
Absolute Error (min) 325 = 0.001111

Diff. Non. Lin. Error (max) 40 = 0.003747
Diff. Non. Lin. Error (min) FF = -0.001071

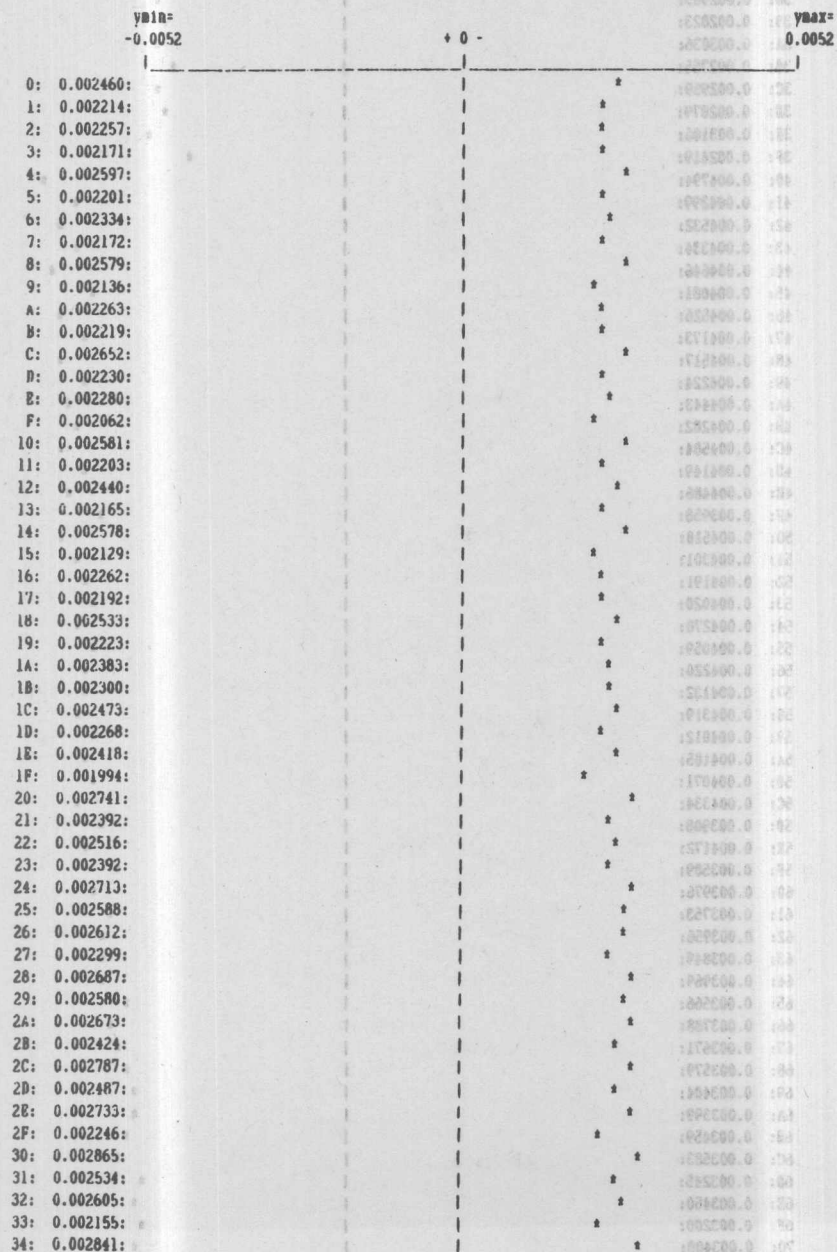
Term. Non. Lin. Error (max) 325 = -0.004102
Term. Non. Lin. Error (min) 40 = 0.002148

Maximum Reliability Error 3D1 = 0.001875
Minimum Reliability Error 3A7 = 0.000974

Resolution is 1024 levels.

```

Absolute Error, SN = 4130



05-180015

270365-69

(bound) Absolute Error, SN = 4130

\$200.00

[illegible]

270365-70

6-228-3

6-229-3

AD: 0.001592:	100500.0	100
AE: 0.001781:	100500.0	100
AF: 0.001538:	100500.0	100
AG: 0.001906:	100500.0	100
AH: 0.001724:	100500.0	100
AI: 0.001887:	100500.0	100
AJ: 0.001773:	100500.0	100
AK: 0.001585:	100500.0	100
AL: 0.001598:	100500.0	100
AM: 0.001650:	100500.0	100
AN: 0.001554:	100500.0	100
AO: 0.001715:	100500.0	100
AP: 0.001545:	100500.0	100
AQ: 0.001653:	100500.0	100
AR: 0.001474:	100500.0	100
AS: 0.001467:	100500.0	100
AT: 0.001384:	100500.0	100
AU: 0.001588:	100500.0	100
AV: 0.001028:	100500.0	100
AW: 0.003214:	100500.0	100
AX: 0.002914:	100500.0	100
AY: 0.002966:	100500.0	100
AZ: 0.002779:	100500.0	100
BA: 0.003087:	100500.0	100
BB: 0.002717:	100500.0	100
BC: 0.003096:	100500.0	100
BD: 0.002806:	100500.0	100
BE: 0.003030:	100500.0	100
BF: 0.002796:	100500.0	100
BG: 0.002642:	100500.0	100
BH: 0.002885:	100500.0	100
BI: 0.003040:	100500.0	100
BJ: 0.002719:	100500.0	100
BK: 0.002878:	100500.0	100
BL: 0.002742:	100500.0	100
BM: 0.002845:	100500.0	100
BN: 0.002546:	100500.0	100
BO: 0.002790:	100500.0	100
BP: 0.002395:	100500.0	100
BQ: 0.002848:	100500.0	100
BR: 0.002487:	100500.0	100
BS: 0.002768:	100500.0	100
BT: 0.002700:	100500.0	100
BU: 0.002681:	100500.0	100
BV: 0.002617:	100500.0	100
BW: 0.002755:	100500.0	100
BX: 0.002643:	100500.0	100
BY: 0.002684:	100500.0	100
BZ: 0.002398:	100500.0	100
CA: 0.002553:	100500.0	100
CB: 0.002223:	100500.0	100
CC: 0.002483:	100500.0	100
CD: 0.001878:	100500.0	100
CE: 0.002439:	100500.0	100
CF: 0.002206:	100500.0	100
CG: 0.002083:	100500.0	100
CH: 0.002055:	100500.0	100
CI: 0.002288:	100500.0	100
CJ: 0.002144:	100500.0	100
CK: 0.002356:	100500.0	100

270365-72

Absolute Error, SN = 4130 (Continued)

E9:	0.002225:		:827000.0	:851
EA:	0.002263:		:600000.0	:851
EB:	0.002113:		:800000.0	:851
EC:	0.002233:		:000100.0	:851
ED:	0.002172:		:700000.0	:851
EE:	0.002369:		:101100.0	:851
EF:	0.002149:		:010100.0	:851
F0:	0.002216:		:000000.0	:851
F1:	0.001841:		:500000.0	:851
F2:	0.002051:		:510000.0	:851
F3:	0.001935:		:610000.0	:851
F4:	0.001965:		:600000.0	:851
F5:	0.001729:		:170000.0	:851
F6:	0.001979:		:110000.0	:851
F7:	0.001899:		:100000.0	:851
F8:	0.001589:		:950000.0	:851
F9:	0.001718:		:700000.0	:851
FA:	0.001935:		:000000.0	:851
FB:	0.001756:		:000000.0	:851
FC:	0.001975:		:700100.0	:851
FD:	0.001832:		:020000.0	:851
FE:	0.001920:		:000000.0	:851
FF:	0.001041:		:000000.0	:851
100:	0.002291:		:500100.0	:851
101:	0.002008:		:000000.0	:851
102:	0.002296:		:030000.0	:851
103:	0.001975:		:100000.0	:851
104:	0.001946:		:015000.0	:851
105:	0.001874:		:001500.0	:851
106:	0.001884:		:750500.0	:851
107:	0.001817:		:001100.0	:851
108:	0.002135:		:700500.0	:851
109:	0.001921:		:700500.0	:851
10A:	0.002009:		:000500.0	:851
10B:	0.001832:		:000500.0	:851
10C:	0.001903:		:020500.0	:851
10D:	0.001694:		:000500.0	:851
10E:	0.001838:		:000500.0	:851
10F:	0.001537:		:500500.0	:851
110:	0.001681:		:000500.0	:851
111:	0.001436:		:000500.0	:851
112:	0.001730:		:000500.0	:851
113:	0.001631:		:700500.0	:851
114:	0.001636:		:700500.0	:851
115:	0.001374:		:750500.0	:851
116:	0.001550:		:510500.0	:851
117:	0.001500:		:500500.0	:851
118:	0.001530:		:000500.0	:851
119:	0.001411:		:070500.0	:851
11A:	0.001390:		:000500.0	:851
11B:	0.001271:		:000500.0	:851
11C:	0.001321:		:000500.0	:851
11D:	0.001074:		:000500.0	:851
11E:	0.001268:		:170500.0	:851
11F:	0.000814:		:000500.0	:851
120:	0.001401:		:001500.0	:851
121:	0.001052:		:000500.0	:851
122:	0.001193:		:000500.0	:851
123:	0.001106:		:000500.0	:851
124:	0.001253:		:000500.0	:851

NT-288055

270365-73

Absolute Error, SN = 4130 (Continued)

125: 0.000758:		:#55500.0	:02
126: 0.000953:		:#A5500.0	:02
127: 0.000976:		:#55500.0	:02
128: 0.001080:		:#55500.0	:02
129: 0.000937:		:#55500.0	:02
12A: 0.001181:		:#00500.0	:02
12B: 0.001018:		:#05500.0	:02
12C: 0.000959:		:#55500.0	:02
12D: 0.000862:		:#0100.0	:02
12E: 0.000812:		:#20500.0	:02
12F: 0.000813:		:#05100.0	:02
130: 0.000933:		:#0100.0	:02
131: 0.000671:		:#555100.0	:02
132: 0.000811:		:#555100.0	:02
133: 0.000634:		:#000100.0	:02
134: 0.000929:		:#05100.0	:02
135: -0.000647:		:#555100.0	:02
136: 0.000880:		:#555100.0	:02
137: 0.000539:		:#055100.0	:02
138: 0.001027:		:#555100.0	:02
139: 0.000850:		:#555100.0	:02
13A: 0.000749:		:#555100.0	:02
13B: 0.000809:		:#555100.0	:02
13C: 0.001032:		:#55500.0	:02
13D: 0.000788:		:#000500.0	:02
13E: 0.000963:		:#55500.0	:02
13F: -0.000681:		:#555100.0	:02
140: 0.002218:		:#000100.0	:02
141: 0.002186:		:#555100.0	:02
142: 0.002327:		:#000100.0	:02
143: 0.002196:		:#555100.0	:02
144: 0.002447:		:#05500.0	:02
145: 0.002267:		:#555100.0	:02
146: 0.002435:		:#55500.0	:02
147: 0.002385:		:#555100.0	:02
148: 0.002554:		:#555100.0	:02
149: 0.002284:		:#000100.0	:02
14A: 0.002420:		:#555100.0	:02
14B: 0.002482:		:#555100.0	:02
14C: 0.002523:		:#555100.0	:02
14D: 0.002299:		:#555100.0	:02
14E: 0.002303:		:#555100.0	:02
14F: 0.002097:		:#555100.0	:02
150: 0.002267:		:#555100.0	:02
151: 0.002127:		:#555100.0	:02
152: 0.002312:		:#555100.0	:02
153: 0.002092:		:#555100.0	:02
154: 0.002264:		:#555100.0	:02
155: 0.001976:		:#555100.0	:02
156: 0.002034:		:#555100.0	:02
157: 0.002084:		:#555100.0	:02
158: 0.002235:		:#555100.0	:02
159: 0.001959:		:#555100.0	:02
15A: 0.002071:		:#555100.0	:02
15B: 0.002048:		:#55500.0	:02
15C: 0.002104:		:#555100.0	:02
15D: 0.001990:		:#555100.0	:02
15E: 0.002110:		:#555100.0	:02
15F: 0.001935:		:#555100.0	:02
160: 0.002075:		:#555100.0	:02

87-000073

270365-74

Absolute Error, SN = 4130 (Continued)

161:	0.001755:	1585100.0	1601
162:	0.001922:	1585100.0	1601
163:	0.001706:	1585100.0	1601
164:	0.001984:	1585100.0	1601
165:	0.001481:	1585100.0	1601
166:	0.001830:	1585100.0	1601
167:	0.001812:	1585100.0	1601
168:	0.001987:	1585100.0	1601
169:	0.001880:	1585100.0	1601
16A:	0.002022:	1585100.0	1601
16B:	0.001736:	1585100.0	1601
16C:	0.001873:	1585100.0	1601
16D:	0.001595:	1585100.0	1601
16E:	0.001620:	1585100.0	1601
16F:	0.001649:	1585100.0	1601
170:	0.001770:	1585100.0	1601
171:	0.001492:	1585100.0	1601
172:	0.001635:	1585100.0	1601
173:	0.001572:	1585100.0	1601
174:	0.001725:	1585100.0	1601
175:	0.001534:	1585100.0	1601
176:	0.001601:	1585100.0	1601
177:	0.001527:	1585100.0	1601
178:	0.001743:	1585100.0	1601
179:	0.001443:	1585100.0	1601
17A:	0.001623:	1585100.0	1601
17B:	0.001578:	1585100.0	1601
17C:	0.001528:	1585100.0	1601
17D:	0.001386:	1585100.0	1601
17E:	0.001466:	1585100.0	1601
17F:	0.001457:	1585100.0	1601
180:	0.001971:	1585100.0	1601
181:	0.001741:	1585100.0	1601
182:	0.001816:	1585100.0	1601
183:	0.001707:	1585100.0	1601
184:	0.001894:	1585100.0	1601
185:	0.001598:	1585100.0	1601
186:	0.001600:	1585100.0	1601
187:	0.001498:	1585100.0	1601
188:	0.001771:	1585100.0	1601
189:	0.001478:	1585100.0	1601
18A:	0.001654:	1585100.0	1601
18B:	0.001591:	1585100.0	1601
18C:	0.001732:	1585100.0	1601
18D:	0.001404:	1585100.0	1601
18E:	0.001536:	1585100.0	1601
18F:	0.001411:	1585100.0	1601
190:	0.001811:	1585100.0	1601
191:	0.001467:	1585100.0	1601
192:	0.001372:	1585100.0	1601
193:	0.001370:	1585100.0	1601
194:	0.001323:	1585100.0	1601
195:	0.001306:	1585100.0	1601
196:	0.001429:	1585100.0	1601
197:	0.001025:	1585100.0	1601
198:	0.001585:	1585100.0	1601
199:	0.001281:	1585100.0	1601
19A:	0.001465:	1585100.0	1601
19B:	0.001323:	1585100.0	1601
19C:	0.001540:	1585100.0	1601

ST-282013

270365-75

Absolute Error, SN = 4130 (Continued)

19D: 0.001262:	:20T100.0 :181
19E: 0.001245:	:559100.0 :531
19F: 0.001201:	:30T100.0 :121
1A0: 0.001413:	:302100.0 :131
1A1: 0.001170:	:183100.0 :231
1A2: 0.001361:	:003100.0 :101
1A3: 0.001321:	:510100.0 :171
1A4: 0.001181:	:702100.0 :131
1A5: 0.000872:	:002100.0 :101
1A6: 0.001086:	:552100.0 :131
1A7: 0.001080:	:30T100.0 :131
1A8: 0.001195:	:6T0100.0 :531
1A9: 0.001138:	:202100.0 :101
1AA: 0.001204:	:033100.0 :531
1AB: 0.001230:	:933100.0 :131
1AC: 0.001210:	:00T100.0 :10T1
1AD: 0.000971:	:591100.0 :1T1
1AE: 0.001083:	:223100.0 :1T1
1AF: 0.001274:	:5T1100.0 :1T1
1B0: 0.001211:	:23T100.0 :1T1
1B1: 0.001133:	:02T100.0 :1T1
1B2: 0.001069:	:103100.0 :10T1
1B3: 0.001095:	:T32100.0 :1T1
1B4: 0.001065:	:C1100.0 :10T1
1B5: 0.001081:	:C1100.0 :10T1
1B6: 0.001124:	:C53100.0 :10T1
1B7: 0.001079:	:0T3100.0 :10T1
1B8: 0.001040:	:852100.0 :10T1
1B9: 0.001081:	:301100.0 :10T1
1BA: 0.001183:	:00T100.0 :10T1
1BB: 0.001297:	:T0T100.0 :10T1
1BC: 0.001124:	:1T0100.0 :10T1
1BD: 0.001006:	:310100.0 :131
1BE: 0.001046:	:010100.0 :131
1BF: 0.001061:	:T0100.0 :10T1
1C0: 0.002475:	:300100.0 :10T1
1C1: 0.002358:	:002100.0 :10T1
1C2: 0.002538:	:003100.0 :10T1
1C3: 0.002457:	:003100.0 :10T1
1C4: 0.002712:	:T0T100.0 :10T1
1C5: 0.002415:	:0T0100.0 :10T1
1C6: 0.002579:	:023100.0 :10T1
1C7: 0.002436:	:102100.0 :10T1
1C8: 0.002796:	:52T100.0 :10T1
1C9: 0.002388:	:103100.0 :10T1
1CA: 0.002368:	:002100.0 :10T1
1CB: 0.002426:	:113100.0 :10T1
1CC: 0.002661:	:110100.0 :10T1
1CD: 0.002462:	:T33100.0 :10T1
1CE: 0.002497:	:5T2100.0 :10T1
1CF: 0.002396:	:0T2100.0 :10T1
1D0: 0.002617:	:C3C100.0 :10T1
1D1: 0.002399:	:30C100.0 :10T1
1D2: 0.002503:	:T53100.0 :10T1
1D3: 0.002453:	:203100.0 :10T1
1D4: 0.002623:	:202100.0 :10T1
1D5: 0.002414:	:105100.0 :10T1
1D6: 0.002423:	:203100.0 :10T1
1D7: 0.002490:	:C3C100.0 :10T1
1D8: 0.002606:	:002100.0 :10T1

37-388075

270365-76

Absolute Error, SN = 4130 (Continued)

1D9: 0.002351:	:50100.0 :21
1DA: 0.002439:	:50100.0 :21
1DB: 0.002382:	:50100.0 :21
1DC: 0.002426:	:50100.0 :21
1DD: 0.002376:	:50100.0 :21
1DE: 0.002443:	:50100.0 :21
1DF: 0.002531:	:50100.0 :21
1E0: 0.002583:	:50100.0 :21
1E1: 0.002030:	:50100.0 :21
1E2: 0.002371:	:50100.0 :21
1E3: 0.002043:	:50100.0 :21
1E4: 0.002350:	:50100.0 :21
1E5: 0.002166:	:50100.0 :21
1E6: 0.002351:	:50100.0 :21
1E7: 0.002363:	:50100.0 :21
1E8: 0.002455:	:50100.0 :21
1E9: 0.002002:	:50100.0 :21
1EA: 0.002299:	:50100.0 :21
1EB: 0.002146:	:50100.0 :21
1EC: 0.002279:	:50100.0 :21
1ED: 0.002072:	:50100.0 :21
1EE: 0.001960:	:50100.0 :21
1EF: 0.002221:	:50100.0 :21
1F0: 0.002314:	:50100.0 :21
1F1: 0.001940:	:50100.0 :21
1F2: 0.002086:	:50100.0 :21
1F3: 0.002310:	:50100.0 :21
1F4: 0.002188:	:50100.0 :21
1F5: 0.002075:	:50100.0 :21
1F6: 0.002065:	:50100.0 :21
1F7: 0.002267:	:50100.0 :21
1F8: 0.002187:	:50100.0 :21
1F9: 0.002002:	:50100.0 :21
1FA: 0.002120:	:50100.0 :21
1FB: 0.002133:	:50100.0 :21
1FC: 0.002158:	:50100.0 :21
1FD: 0.001937:	:50100.0 :21
1FE: 0.002079:	:50100.0 :21
1FF: 0.001409:	:50100.0 :21
200: 0.001879:	:50100.0 :21
201: 0.001707:	:50100.0 :21
202: 0.001905:	:50100.0 :21
203: 0.001557:	:50100.0 :21
204: 0.001650:	:50100.0 :21
205: 0.001661:	:50100.0 :21
206: 0.001683:	:50100.0 :21
207: 0.001595:	:50100.0 :21
208: 0.001535:	:50100.0 :21
209: 0.001179:	:50100.0 :21
20A: 0.001610:	:50100.0 :21
20B: 0.001454:	:50100.0 :21
20C: 0.001370:	:50100.0 :21
20D: 0.001262:	:50100.0 :21
20E: 0.001179:	:50100.0 :21
20F: 0.000983:	:50100.0 :21
210: 0.001405:	:50100.0 :21
211: 0.001074:	:50100.0 :21
212: 0.001168:	:50100.0 :21
213: 0.001193:	:50100.0 :21
214: 0.001420:	:50100.0 :21

87-200000

270365-77

Absolute Error, SN = 4130 (Continued)

215:	0.001162:	120000.0	1001
216:	0.001323:	100000.0	1001
217:	0.001268:	100000.0	1001
218:	0.001296:	100000.0	1001
219:	0.001147:	100000.0	1001
21A:	0.001036:	100000.0	1001
21B:	0.001170:	100000.0	1001
21C:	0.001551:	100000.0	1001
21D:	0.001065:	100000.0	1001
21E:	0.001216:	100000.0	1001
21F:	0.000666:	100000.0	1001
220:	0.001304:	100000.0	1001
221:	0.000988:	100000.0	1001
222:	0.001207:	100000.0	1001
223:	0.001066:	100000.0	1001
224:	0.001079:	100000.0	1001
225:	0.001029:	100000.0	1001
226:	0.000971:	100000.0	1001
227:	0.000968:	100000.0	1001
228:	0.001203:	100000.0	1001
229:	0.000949:	100000.0	1001
22A:	0.001026:	100000.0	1001
22B:	0.001051:	100000.0	1001
22C:	0.001118:	100000.0	1001
22D:	0.000887:	100000.0	1001
22E:	0.001149:	100000.0	1001
22F:	0.000738:	100000.0	1001
230:	0.001214:	100000.0	1001
231:	0.000920:	100000.0	1001
232:	0.001203:	100000.0	1001
233:	0.000978:	100000.0	1001
234:	0.001203:	100000.0	1001
235:	0.001081:	100000.0	1001
236:	0.001003:	100000.0	1001
237:	0.001053:	100000.0	1001
238:	0.001235:	100000.0	1001
239:	0.000705:	100000.0	1001
23A:	0.001066:	100000.0	1001
23B:	0.000924:	100000.0	1001
23C:	0.001087:	100000.0	1001
23D:	0.001000:	100000.0	1001
23E:	0.001006:	100000.0	1001
23F:	0.000785:	100000.0	1001
240:	0.002137:	100000.0	1001
241:	0.001968:	100000.0	1001
242:	0.002196:	100000.0	1001
243:	0.002027:	100000.0	1001
244:	0.002162:	100000.0	1001
245:	0.001918:	100000.0	1001
246:	0.002075:	100000.0	1001
247:	0.001871:	100000.0	1001
248:	0.002060:	100000.0	1001
249:	0.002108:	100000.0	1001
24A:	0.002100:	100000.0	1001
24B:	0.002060:	100000.0	1001
24C:	0.002217:	100000.0	1001
24D:	0.002035:	100000.0	1001
24E:	0.002245:	100000.0	1001
24F:	0.002190:	100000.0	1001
250:	0.002415:	100000.0	1001

TT-882015

270365-78

Absolute Error, SN = 4130 (Continued)

251:	0.002013:		:050100.0	*:085
252:	0.002259:		:002100.0	:085
253:	0.002068:		:002100.0	:085
254:	0.002370:		:002100.0	:085
255:	0.002213:		:050100.0	:085
256:	0.002314:		:050100.0	:085
257:	0.002207:		:050100.0	:085
258:	0.002259:		:050100.0	:085
259:	0.002090:		:050100.0	:085
25A:	0.001956:		:050100.0	:085
25B:	0.002095:		:050100.0	:085
25C:	0.002377:		:002100.0	:085
25D:	0.002086:		:050100.0	:085
25E:	0.002090:		:002100.0	:085
25F:	0.001972:		:050100.0	:085
260:	0.002137:		:050100.0	:085
261:	0.001808:		:002000.0	*:085
262:	0.002022:		:002100.0	:085
263:	0.001944:		:002000.0	:085
264:	0.002053:		:002000.0	:085
265:	0.001856:		:002000.0	:085
266:	0.002042:		:050100.0	:085
267:	0.001940:		:050100.0	:085
268:	0.002020:		:050100.0	:085
269:	0.001762:		:002000.0	*:085
26A:	0.001820:		:050100.0	:085
26B:	0.001773:		:002000.0	:085
26C:	0.001850:		:050100.0	:085
26D:	0.001685:		:050100.0	:085
26E:	0.001910:		:050100.0	:085
26F:	0.001794:		:050100.0	:085
270:	0.001748:		:050100.0	:085
271:	0.001653:		:050100.0	:085
272:	0.001632:		:050100.0	:085
273:	0.001540:		:050100.0	:085
274:	0.001677:		:050100.0	:085
275:	0.001356:		:050100.0	:085
276:	0.001582:		:050100.0	:085
277:	0.001630:		:050100.0	:085
278:	0.001505:		:050100.0	:085
279:	0.001403:		:050100.0	:085
27A:	0.001464:		:050100.0	:085
27B:	0.001402:		:050100.0	:085
27C:	0.001620:		:050100.0	:085
27D:	0.001106:		:050100.0	:085
27E:	0.001437:		:050100.0	:085
27F:	0.001276:		:050100.0	:085
280:	0.001913:		:050100.0	:085
281:	0.001950:		:050100.0	:085
282:	0.002095:		:050100.0	:085
283:	0.001620:		:050100.0	:085
284:	0.002096:		:050100.0	:085
285:	0.001850:		:050100.0	:085
286:	0.001951:		:050100.0	:085
287:	0.001836:		:050100.0	:085
288:	0.001726:		:050100.0	:085
289:	0.001690:		:050100.0	:085
28A:	0.001743:		:050100.0	:085
28B:	0.001775:		:050100.0	:085
28C:	0.001551:		:050100.0	:085

08-082075

270365-79

Absolute Error, SN = 4130 (Continued)

28D: 0.001620:		:E10500.0	:125
28E: 0.001599:		:E25500.0	:125
28F: 0.001536:		:E90500.0	:125
290: 0.001558:		:E7E500.0	:125
291: 0.001423:		:E15200.0	:125
292: 0.001437:		:E16500.0	:125
293: 0.001255:		:E05500.0	:125
294: 0.001423:		:E25500.0	:125
295: 0.001151:		:E99500.0	:125
296: 0.001336:		:E20100.0	:125
297: 0.001311:		:E90500.0	:125
298: 0.001308:		:E7E500.0	:125
299: 0.001125:		:E99500.0	:125
29A: 0.001060:		:E00100.0	:125
29B: 0.001134:		:E7F100.0	:125
29C: 0.001209:		:E7E100.0	:125
29D: 0.000856:		:E00100.0	:125
29E: 0.001095:		:E50100.0	:125
29F: 0.000790:		:E07100.0	:125
2A0: 0.000988:		:E20100.0	:125
2A1: 0.000839:		:E0100.0	:125
2A2: 0.001122:		:E30500.0	:125
2A3: 0.000913:		:E01100.0	:125
2A4: 0.000971:		:E05100.0	:125
2A5: 0.000710:		:E0C100.0	:125
2A6: 0.000879:		:E05100.0	:125
2A7: 0.000807:		:E07100.0	:125
2A8: 0.001102:		:E00100.0	:125
2A9: 0.000720:		:E0100.0	:125
2AA: -0.000620:		:E10100.0	:125
2AB: 0.000799:		:E07100.0	:125
2AC: 0.000991:		:E00100.0	:125
2AD: 0.000727:		:E00100.0	:125
2AE: 0.000684:		:E00100.0	:125
2AF: 0.000683:		:E02100.0	:125
2B0: 0.000713:		:E00100.0	:125
2B1: -0.000782:		:E00100.0	:125
2B2: 0.000601:		:E02100.0	:125
2B3: -0.000704:		:E00100.0	:125
2B4: 0.000647:		:E00100.0	:125
2B5: -0.000815:		:E00100.0	:125
2B6: -0.000685:		:E00100.0	:125
2B7: -0.000716:		:E00100.0	:125
2B8: 0.000688:		:E00100.0	:125
2B9: -0.000764:		:E00100.0	:125
2BA: -0.000661:		:E00100.0	:125
2BB: -0.000781:		:E00100.0	:125
2BC: 0.000904:		:E0100.0	:125
2BD: 0.000707:		:E00100.0	:125
2BE: 0.000763:		:E0000.0	:125
2BF: 0.000844:		:E0100.0	:125
2C0: 0.002248:		:E00000.0	:125
2C1: 0.001988:		:E00100.0	:125
2C2: 0.002117:		:E00100.0	:125
2C3: 0.002005:		:E00100.0	:125
2C4: 0.002275:		:E00100.0	:125
2C5: 0.002183:		:E00100.0	:125
2C6: 0.002092:		:E00100.0	:125
2C7: 0.002171:		:E00100.0	:125
2C8: 0.002366:		:E00100.0	:125

GT-282075

270365-80

Absolute Error, SN = 4130 (Continued)

2C9: 0.002105:		:T05100.0 * :89C
2CA: 0.002047:		:T1E100.0 * :89C
2CB: 0.002142:		:T2E100.0 * :89C
2CC: 0.002308:		:T3E100.0 * :89C
2CD: 0.002226:		:T00100.0 * :89C
2CE: 0.002106:		:T05100.0 * :89C
2CF: 0.001931:		:T4C100.0 * :89C
2D0: 0.002298:		:T05100.0 * :89C
2D1: 0.001963:		:T2E100.0 * :89C
2D2: 0.002106:		:T0E100.0 * :89C
2D3: 0.002014:		:T3E000.0 * :89C
2D4: 0.002136:		:T50100.0 * :89C
2D5: 0.001849:		:T60000.0 * :89C
2D6: 0.002152:		:T00000.0 * :89C
2D7: 0.002205:		:T20000.0 * :89C
2D8: 0.002087:		:T40000.0 * :89C
2D9: 0.001866:		:T30000.0 * :89C
2DA: 0.002304:		:T00000.0 * :89C
2DB: 0.002234:		:T40000.0 * :89C
2DC: 0.002308:		:T00000.0 * :89C
2DD: 0.001769:		:T00000.0 * :89C
2DE: 0.002155:		:T50000.0 * :89C
2DF: 0.002034:		:T00000.0 * :89C
2E0: 0.001801:		:T50000.0 * :89C
2E1: 0.001788:		:T00000.0 * :89C
2E2: 0.001813:		:T00000.0 * :89C
2E3: 0.001724:		:T00000.0 * :89C
2E4: 0.001537:		:T00000.0 * :89C
2E5: 0.001622:		:T50000.0 * :89C
2E6: 0.001797:		:T00000.0 * :89C
2E7: 0.001799:		:T00000.0 * :89C
2E8: 0.001720:		:T00000.0 * :89C
2E9: 0.001537:		:T11100.0 * :89C
2EA: 0.001715:		:T00000.0 * :89C
2EB: 0.001385:		:T00000.0 * :89C
2EC: 0.001687:		:T00000.0 * :89C
2ED: 0.001464:		:T00000.0 * :89C
2EE: 0.001508:		:T00000.0 * :89C
2EF: 0.001373:		:T00000.0 * :89C
2F0: 0.001488:		:T00000.0 * :89C
2F1: 0.001379:		:T00000.0 * :89C
2F2: 0.001508:		:T00000.0 * :89C
2F3: 0.001325:		:T00000.0 * :89C
2F4: 0.001385:		:T00000.0 * :89C
2F5: 0.001225:		:T00000.0 * :89C
2F6: 0.001381:		:T00000.0 * :89C
2F7: 0.001301:		:T00000.0 * :89C
2F8: 0.001168:		:T00000.0 * :89C
2F9: 0.001136:		:T00000.0 * :89C
2FA: 0.001032:		:T00000.0 * :89C
2FB: 0.000957:		:T00000.0 * :89C
2FC: 0.001102:		:T00000.0 * :89C
2FD: 0.001088:		:T00000.0 * :89C
2FE: 0.000999:		:T00000.0 * :89C
2FF: 0.001571:		:T50000.0 * :89C
300: 0.001484:		:T00000.0 * :89C
301: 0.001278:		:T00000.0 * :89C
302: 0.001463:		:T00000.0 * :89C
303: 0.001298:		:T00000.0 * :89C
304: 0.001282:		:T00000.0 * :89C

SN-268075

270365-81

Absolute Error, SN = 4130 (Continued)

Absolute Error, SN = 4130 (Continued)

341:	0.002709:	:	:	218100.0	:071 *
342:	0.002828:	:	:	200100.0	:072 *
343:	0.002542:	:	:	200100.0	:073 *
344:	0.002784:	:	:	200500.0	:080 *
345:	0.002719:	:	:	200100.0	:100 *
346:	0.002590:	:	:	200100.0	:101 *
347:	0.002871:	:	:	200100.0	:102 *
348:	0.003014:	:	:	200100.0	:103 *
349:	0.003003:	:	:	200100.0	:104 *
34A:	0.002773:	:	:	200100.0	:105 *
34B:	0.002744:	:	:	200100.0	:106 *
34C:	0.003031:	:	:	200100.0	:107 *
34D:	0.002672:	:	:	200100.0	:108 *
34E:	0.002854:	:	:	200100.0	:109 *
34F:	0.002906:	:	:	200100.0	:110 *
350:	0.002960:	:	:	200100.0	:111 *
351:	0.002742:	:	:	200100.0	:112 *
352:	0.002836:	:	:	200100.0	:113 *
353:	0.002754:	:	:	200100.0	:114 *
354:	0.003072:	:	:	200100.0	:115 *
355:	0.002821:	:	:	200100.0	:116 *
356:	0.003011:	:	:	200100.0	:117 *
357:	0.003037:	:	:	200100.0	:118 *
358:	0.002763:	:	:	200100.0	:119 *
359:	0.002649:	:	:	200100.0	:120 *
35A:	0.002595:	:	:	200100.0	:121 *
35B:	0.002773:	:	:	200100.0	:122 *
35C:	0.002793:	:	:	200100.0	:123 *
35D:	0.002479:	:	:	200100.0	:124 *
35E:	0.002709:	:	:	200100.0	:125 *
35F:	0.002716:	:	:	200100.0	:126 *
360:	0.002505:	:	:	200100.0	:127 *
361:	0.002437:	:	:	200100.0	:128 *
362:	0.002451:	:	:	200100.0	:129 *
363:	0.002320:	:	:	200100.0	:130 *
364:	0.002448:	:	:	200100.0	:131 *
365:	0.002264:	:	:	200100.0	:132 *
366:	0.002375:	:	:	200100.0	:133 *
367:	0.002312:	:	:	200100.0	:134 *
368:	0.002421:	:	:	200100.0	:135 *
369:	0.002251:	:	:	200100.0	:136 *
36A:	0.002330:	:	:	200100.0	:137 *
36B:	0.002272:	:	:	200100.0	:138 *
36C:	0.002269:	:	:	200100.0	:139 *
36D:	0.001925:	:	:	200100.0	:140 *
36E:	0.002158:	:	:	200100.0	:141 *
36F:	0.002229:	:	:	200100.0	:142 *
370:	0.002246:	:	:	200100.0	:143 *
371:	0.001929:	:	:	200100.0	:144 *
372:	0.002095:	:	:	200100.0	:145 *
373:	0.002046:	:	:	200100.0	:146 *
374:	0.002085:	:	:	200100.0	:147 *
375:	0.001876:	:	:	200100.0	:148 *
376:	0.001926:	:	:	200100.0	:149 *
377:	0.002039:	:	:	200100.0	:150 *
378:	0.001967:	:	:	200100.0	:151 *
379:	0.001932:	:	:	200100.0	:152 *
37A:	0.002019:	:	:	200100.0	:153 *
37B:	0.001950:	:	:	200100.0	:154 *
37C:	0.001922:	:	:	200100.0	:155 *

58-286075

270365-83

Absolute Error, SN = 4130 (Continued)

37D: 0.001815:	1801500.0	181
37E: 0.001689:	1801500.0	181
37F: 0.002200:	1801500.0	181
380: 0.002064:	1801500.0	181
381: 0.001764:	1801500.0	181
382: 0.001910:	1801500.0	181
383: 0.001945:	1801500.0	181
384: 0.001913:	1801500.0	181
385: 0.001866:	1801500.0	181
386: 0.001889:	1801500.0	181
387: 0.001800:	1801500.0	181
388: 0.001779:	1801500.0	181
389: 0.001454:	1801500.0	181
38A: 0.001584:	1801500.0	181
38B: 0.001477:	1801500.0	181
38C: 0.001469:	1801500.0	181
38D: 0.001268:	1801500.0	181
38E: 0.001562:	1801500.0	181
38F: 0.001268:	1801500.0	181
390: 0.001568:	1801500.0	181
391: 0.000946:	1801500.0	181
392: 0.001423:	1801500.0	181
393: 0.001232:	1801500.0	181
394: 0.001499:	1801500.0	181
395: 0.001255:	1801500.0	181
396: 0.001087:	1801500.0	181
397: 0.001265:	1801500.0	181
398: 0.001421:	1801500.0	181
399: 0.001169:	1801500.0	181
39A: 0.001269:	1801500.0	181
39B: 0.001245:	1801500.0	181
39C: 0.001440:	1801500.0	181
39D: 0.001153:	1801500.0	181
39E: 0.001402:	1801500.0	181
39F: 0.001260:	1801500.0	181
3A0: 0.001363:	1801500.0	181
3A1: 0.001145:	1801500.0	181
3A2: 0.001221:	1801500.0	181
3A3: 0.001155:	1801500.0	181
3A4: 0.001452:	1801500.0	181
3A5: 0.001302:	1801500.0	181
3A6: 0.001138:	1801500.0	181
3A7: 0.001079:	1801500.0	181
3A8: 0.001378:	1801500.0	181
3A9: 0.001043:	1801500.0	181
3AA: 0.001145:	1801500.0	181
3AB: 0.001207:	1801500.0	181
3AC: 0.001161:	1801500.0	181
3AD: 0.001133:	1801500.0	181
3AE: 0.001137:	1801500.0	181
3AF: 0.001175:	1801500.0	181
3B0: 0.001159:	1801500.0	181
3B1: 0.000747:	1801500.0	181
3B2: 0.000927:	1801500.0	181
3B3: 0.000883:	1801500.0	181
3B4: 0.001127:	1801500.0	181
3B5: 0.000784:	1801500.0	181
3B6: 0.001002:	1801500.0	181
3B7: 0.001058:	1801500.0	181
3B8: 0.000907:	1801500.0	181

00-28075

270365-84

Absolute Error, SN = 4130 (Continued)

Absolute Error, SN = 4130 (Continued)

Non. Lin. Error, SN = 4130

	YMIN = -0.0037	+ 0 -	YMAX = 0.0037
0: -0.000000:		*	
1: -0.000297:		*	
2: -0.000256:		*	
3: -0.000343:		*	
4: 0.000031:		*	
5: -0.000266:		*	
6: -0.000134:		*	
7: -0.000397:		*	
8: 0.000007:		*	
9: -0.000386:		*	
A: -0.000210:		*	
B: -0.000256:		*	
C: 0.000075:		*	
D: -0.000247:		*	
E: -0.000199:		*	
F: -0.000468:		*	
10: -0.000000:		*	
11: -0.000330:		*	
12: -0.000094:		*	
13: -0.000320:		*	
14: 0.000040:		*	
15: -0.000409:		*	
16: -0.000277:		*	
17: -0.000349:		*	
18: -0.000009:		*	
19: -0.000270:		*	
1A: -0.000112:		*	
1B: -0.000296:		*	
1C: -0.000124:		*	
1D: -0.000281:		*	
1E: -0.000132:		*	
1F: -0.000607:		*	
20: 0.000137:		*	
21: -0.000162:		*	
22: -0.000039:		*	
23: -0.000215:		*	
24: 0.000104:		*	
25: 0.000028:		*	
26: 0.000000:		*	
27: -0.000313:		*	
28: 0.000123:		*	
29: -0.000085:		*	
2A: 0.000056:		*	
2B: -0.000143:		*	
2C: 0.000217:		*	
2D: -0.000083:		*	
2E: 0.000061:		*	
2F: -0.000276:		*	
30: 0.000290:		*	
31: 0.000008:		*	
32: 0.000078:		*	
33: -0.000323:		*	
34: 0.000211:		*	

88-382015

270365-30

(Non. Lin. Error, SN = 4130) loadA

35: -0.000065:	* :584000.0 :1V
36: 0.000165:	:570000.0 :5V
37: -0.000406:	* :562000.0 :6V
38: 0.000409:	:554000.0 :8V
39: 0.000186:	:546000.0 :2T
3A: 0.000398:	:538000.0 :8T
3B: 0.000166:	:530000.0 :3T
3C: 0.000368:	:522000.0 :8T
3D: 0.000287:	:514000.0 :8T
3E: 0.000513:	:506000.0 :1AT
3F: -0.000275:	* :498000.0 :8V
40: 0.002147:	:490000.0 :3V
41: 0.001651:	:482000.0 :8T *
42: 0.001983:	:474000.0 :4T *
43: 0.001784:	:466000.0 :8T *
44: 0.001994:	:458000.0 :8T *
45: 0.001478:	:450000.0 :12 *
46: 0.001922:	:442000.0 :50 *
47: 0.001617:	:434000.0 :60 *
48: 0.001910:	:426000.0 :14 *
49: 0.001616:	:418000.0 :40 *
4A: 0.001833:	:410000.0 :120 *
4B: 0.001621:	:402000.0 :12 *
4C: 0.001872:	:394000.0 :80 *
4D: 0.001585:	:386000.0 :18 *
4E: 0.001771:	:378000.0 :16 *
4F: 0.001392:	:370000.0 :32 *
50: 0.001900:	:362000.0 :38 *
51: 0.001682:	:354000.0 :48 *
52: 0.001571:	:346000.0 :138 *
53: 0.001498:	:338000.0 :140 *
54: 0.001755:	:330000.0 :80 *
55: 0.001485:	:322000.0 :118 *
56: 0.001594:	:314000.0 :150 *
57: 0.001455:	:306000.0 :160 *
58: 0.001641:	:298000.0 :120 *
59: 0.001382:	:290000.0 :12 *
5A: 0.001604:	:282000.0 :100 *
5B: 0.001489:	:274000.0 :170 *
5C: 0.001650:	:266000.0 :180 *
5D: 0.001323:	:258000.0 :1 *
5E: 0.001536:	:250000.0 :140 *
5F: 0.000952:	:242000.0 :100 *
60: 0.001437:	:234000.0 :170 *
61: 0.001163:	:226000.0 :1 *
62: 0.001365:	:218000.0 :18 *
63: 0.001156:	:210000.0 :1 *
64: 0.001275:	:202000.0 :1 *
65: 0.000971:	:194000.0 :1 *
66: 0.001141:	:186000.0 :1 *
67: 0.000923:	:178000.0 :1 *
68: 0.000980:	:170000.0 :1 *
69: 0.000803:	:162000.0 :1 *
6A: 0.000797:	:154000.0 :1 *
6B: 0.000806:	:146000.0 :1 *
6C: 0.000928:	:138000.0 :1 *
6D: 0.000589:	:130000.0 :1 *
6E: 0.000793:	:122000.0 :1 *
6F: 0.000592:	:114000.0 :1 *
70: 0.000798:	:106000.0 :1 *

26-885075

270365-34

Non. Lin. Error, SN = 4130 (Continued)

71: 0.000442:	:200000.0 :2E
72: 0.000576:	:201000.0 :2C
73: 0.000537:	:201000.0 :2E
74: 0.000616:	:200000.0 :2C
75: 0.000216:	:201000.0 :2E
76: 0.000493:	:200000.0 :2C
77: 0.000393:	:200000.0 :2C
78: 0.000330:	:200000.0 :2C
79: 0.000111:	:200000.0 :2C
7A: 0.000216:	:200000.0 :2C
7B: 0.000158:	:200000.0 :2C
7C: 0.000148:	:201000.0 :2E
7D: -0.000060:	* :200000.0 :2C
7E: 0.000081:	:200000.0 :2C
7F: -0.000601:	* :200000.0 :2C
80: 0.000691:	:200000.0 :2C
81: 0.000447:	:200000.0 :2C
82: 0.000588:	:200000.0 :2C
83: 0.000434:	:200000.0 :2C
84: 0.000566:	:200000.0 :2C
85: 0.000265:	:200000.0 :2C
86: 0.000397:	:200000.0 :2C
87: 0.000157:	:200000.0 :2C
88: 0.000396:	:200000.0 :2C
89: -0.000196:	* :200000.0 :2C
8A: 0.000339:	:200000.0 :2C
8B: -0.000021:	* :200000.0 :2C
8C: 0.000166:	:200000.0 :2C
8D: -0.000104:	* :200000.0 :2C
8E: -0.000163:	* :200000.0 :2C
8F: -0.000285:	* :200000.0 :2C
90: 0.000089:	:200000.0 :2C
91: -0.000055:	* :200000.0 :2C
92: 0.000057:	:200000.0 :2C
93: -0.000129:	* :200000.0 :2C
94: 0.000025:	* :200000.0 :2C
95: -0.000194:	* :200000.0 :2C
96: -0.000048:	* :200000.0 :2C
97: -0.000255:	* :200000.0 :2C
98: -0.000119:	* :200000.0 :2C
99: -0.000445:	* :200000.0 :2C
9A: -0.000214:	* :200000.0 :2C
9B: -0.000376:	* :200000.0 :2C
9C: -0.000305:	* :200000.0 :2C
9D: -0.000650:	* :200000.0 :2C
9E: -0.000467:	* :200000.0 :2C
9F: -0.000967:	* :200000.0 :2C
A0: -0.000481:	* :200000.0 :2C
A1: -0.000830:	* :200000.0 :2C
A2: -0.000416:	* :200000.0 :2C
A3: -0.000790:	* :200000.0 :2C
A4: -0.000574:	* :200000.0 :2C
A5: -0.000848:	* :200000.0 :2C
A6: -0.000709:	* :200000.0 :2C
A7: -0.000898:	* :200000.0 :2C
A8: -0.000774:	* :200000.0 :2C
A9: -0.000892:	* :200000.0 :2C
AA: -0.000768:	* :200000.0 :2C
AB: -0.000911:	* :200000.0 :2C
AC: -0.000824:	* :200000.0 :2C

*C-288075

270365-35

Non. Lin. Error, SN = 4130 (Continued)

AD: -0.001097:	* 00000.0- 112
AE: -0.000960:	* 00000.0- 112
AF: -0.001154:	* 00000.0- 112
BD: -0.000787:	* 00000.0- 112
BI: -0.001021:	* 00000.0- 112
BZ: -0.000909:	* 00000.0- 112
B3: -0.001024:	* 00000.0- 112
B4: -0.001064:	* 00000.0- 112
B5: -0.001152:	* 00000.0- 112
B6: -0.001051:	* 00000.0- 112
B7: -0.001199:	* 00000.0- 112
B8: -0.001089:	* 00000.0- 112
B9: -0.001260:	* 00000.0- 112
BA: -0.001104:	* 00000.0- 112
BB: -0.001284:	* 00000.0- 112
BC: -0.001242:	* 00000.0- 112
BD: -0.001376:	* 00000.0- 112
BE: -0.001174:	* 00000.0- 112
BF: -0.001735:	* 00000.0- 112
C0: 0.000398:	* 00000.0- 112
C1: 0.000097:	* 00000.0- 112
C2: 0.000248:	* 00000.0- 112
C3: 0.000109:	* 00000.0- 112
C4: 0.000316:	* 00000.0- 112
C5: -0.000054:	* 00000.0- 112
C6: 0.000322:	* 00000.0- 112
C7: -0.000018:	* 00000.0- 112
C8: 0.000254:	* 00000.0- 112
C9: 0.000018:	* 00000.0- 112
CA: -0.000086:	* 00000.0- 112
CB: 0.000005:	* 00000.0- 112
CC: 0.000208:	* 00000.0- 112
CD: -0.000113:	* 00000.0- 112
CE: 0.000094:	* 00000.0- 112
CF: -0.000093:	* 00000.0- 112
D0: 0.000058:	* 00000.0- 112
D1: -0.000191:	* 00000.0- 112
D2: -0.000049:	* 00000.0- 112
D3: -0.000445:	* 00000.0- 112
D4: 0.000056:	* 00000.0- 112
D5: -0.000306:	* 00000.0- 112
D6: 0.000023:	* 00000.0- 112
D7: -0.000145:	* 00000.0- 112
D8: -0.000116:	* 00000.0- 112
D9: -0.000231:	* 00000.0- 112
DA: -0.000044:	* 00000.0- 112
DB: -0.000158:	* 00000.0- 112
DC: -0.000168:	* 00000.0- 112
DD: -0.000455:	* 00000.0- 112
DE: -0.000301:	* 00000.0- 112
DF: -0.000633:	* 00000.0- 112
E0: -0.000324:	* 00000.0- 112
E1: -0.000830:	* 00000.0- 112
E2: -0.000421:	* 00000.0- 112
E3: -0.000605:	* 00000.0- 112
E4: -0.000729:	* 00000.0- 112
E5: -0.000709:	* 00000.0- 112
E6: -0.000527:	* 00000.0- 112
E7: -0.000672:	* 00000.0- 112
E8: -0.000462:	* 00000.0- 112

TC-88005

270365-36

Non. Lin. Error, SN = 4130 (Continued)

[illegible]

125: -0.002140:	*	:875100.0- :11
126: -0.001946:	*	:820100.0- :54
127: -0.001975:	*	:875100.0- :04
128: -0.001772:	*	:800100.0- :04
129: -0.001967:	*	:500100.0- :02
12A: -0.001774:	*	:821100.0- :02
12B: -0.001888:	*	:875100.0- :01
12C: -0.001948:	*	:001100.0- :02
12D: -0.002097:	*	:025100.0- :01
12E: -0.002048:	*	:000000.0- :01
12F: -0.002148:	*	:005100.0- :00
130: -0.001980:	*	:071100.0- :03
131: -0.002243:	*	:000100.0- :00
132: -0.002054:	*	:875100.0- :02
133: -0.002233:	*	:070100.0- :04
134: -0.002039:	*	:075100.0- :00
135: -0.002342:	*	:072100.0- :01
136: -0.002083:	*	:200100.0- :05
137: -0.002333:	*	:000100.0- :00
138: -0.001896:	*	:030100.0- :01
139: -0.002125:	*	:002100.0- :02
13A: -0.002177:	*	:020100.0- :01
13B: -0.002168:	*	:000100.0- :01
13C: -0.001897:	*	:200100.0- :01
13D: -0.002142:	*	:070100.0- :00
13E: -0.002018:	*	:000100.0- :01
13F: -0.002490:	*	:000100.0- :01
140: -0.000666:	*	:000100.0- :01
141: -0.000700:	*	:070100.0- :00
142: -0.000560:	*	:000100.0- :01
143: -0.000692:	*	:110100.0- :01
144: -0.000493:	*	:000100.0- :00
145: -0.000724:	*	:070100.0- :01
146: -0.000607:	*	:000100.0- :01
147: -0.000659:	*	:000100.0- :00
148: -0.000441:	*	:000100.0- :01
149: -0.000712:	*	:070100.0- :01
14A: -0.000578:	*	:070100.0- :01
14B: -0.000517:	*	:000100.0- :01
14C: -0.000527:	*	:000100.0- :01
14D: -0.000753:	*	:000100.0- :01
14E: -0.000650:	*	:070100.0- :01
14F: -0.000857:	*	:020100.0- :01
150: -0.000688:	*	:020100.0- :01
151: -0.000880:	*	:000100.0- :01
152: -0.000746:	*	:000100.0- :01
153: -0.000917:	*	:070100.0- :01
154: -0.000747:	*	:000100.0- :01
155: -0.000986:	*	:000100.0- :01
156: -0.000929:	*	:100100.0- :01
157: -0.000931:	*	:000100.0- :01
158: -0.000731:	*	:070100.0- :01
159: -0.001008:	*	:000100.0- :01
15A: -0.000898:	*	:000100.0- :01
15B: -0.000972:	*	:000100.0- :01
15C: -0.000867:	*	:000100.0- :01
15D: -0.001075:	*	:000100.0- :01
15E: -0.000914:	*	:000100.0- :01
15F: -0.001140:	*	:000100.0- :01
160: -0.001002:	*	:000100.0- :01

98-880073

270365-38

Non. Lin. Error, SN = 4130 (Continued)

161: -0.001273:		* 104100.0- 101
162: -0.001057:		* 104100.0- 101
163: -0.001275:		* 104100.0- 101
164: -0.001048:		* 104100.0- 101
165: -0.001502:		* 104100.0- 101
166: -0.001155:		* 104100.0- 101
167: -0.001274:		* 104100.0- 101
168: -0.001100:		* 104100.0- 101
169: -0.001259:		* 104100.0- 101
16A: -0.000968:		* 104100.0- 101
16B: -0.001205:		* 104100.0- 101
16C: -0.001170:		* 104100.0- 101
16D: -0.001449:		* 104100.0- 101
16E: -0.001375:		* 104100.0- 101
16F: -0.001347:		* 104100.0- 101
170: -0.001278:		* 104100.0- 101
171: -0.001557:		* 104100.0- 101
172: -0.001365:		* 104100.0- 101
173: -0.001430:		* 104100.0- 101
174: -0.001328:		* 104100.0- 101
175: -0.001520:		* 104100.0- 101
176: -0.001455:		* 104100.0- 101
177: -0.001480:		* 104100.0- 101
178: -0.001315:		* 104100.0- 101
179: -0.001617:		* 104100.0- 101
17A: -0.001338:		* 104100.0- 101
17B: -0.001484:		* 104100.0- 101
17C: -0.001486:		* 104100.0- 101
17D: -0.001679:		* 104100.0- 101
17E: -0.001500:		* 104100.0- 101
17F: -0.001611:		* 104100.0- 101
180: -0.001098:		* 104100.0- 101
181: -0.001379:		* 104100.0- 101
182: -0.001306:		* 104100.0- 101
183: -0.001366:		* 104100.0- 101
184: -0.001230:		* 104100.0- 101
185: -0.001478:		* 104100.0- 101
186: -0.001377:		* 104100.0- 101
187: -0.001480:		* 104100.0- 101
188: -0.001309:		* 104100.0- 101
189: -0.001553:		* 104100.0- 101
18A: -0.001378:		* 104100.0- 101
18B: -0.001493:		* 104100.0- 101
18C: -0.001353:		* 104100.0- 101
18D: -0.001682:		* 104100.0- 101
18E: -0.001502:		* 104100.0- 101
18F: -0.001678:		* 104100.0- 101
190: -0.001229:		* 104100.0- 101
191: -0.001624:		* 104100.0- 101
192: -0.001671:		* 104100.0- 101
193: -0.001674:		* 104100.0- 101
194: -0.001672:		* 104100.0- 101
195: -0.001841:		* 104100.0- 101
196: -0.001669:		* 104100.0- 101
197: -0.002024:		* 104100.0- 101
198: -0.001466:		* 104100.0- 101
199: -0.001871:		* 104100.0- 101
19A: -0.001688:		* 104100.0- 101
19B: -0.001782:		* 104100.0- 101
19C: -0.001516:		* 104100.0- 101

88-2800'S

270365-39

Non. Lin. Error, SN = 4130 (Continued)

19D: -0.001845:	*	:00000.0- :00
19E: -0.001864:	*	:00000.0- :00
19F: -0.001909:	*	:00000.0- :00
1A0: -0.001698:	*	:20000.0- :00
1A1: -0.001943:	*	:00000.0- :00
1A2: -0.001803:	*	:00000.0- :00
1A3: -0.001894:	*	:00000.0- :00
1A4: -0.001936:	*	:00000.0- :00
1A5: -0.002146:	*	:00000.0- :00
1A6: -0.001983:	*	:00000.0- :00
1A7: -0.002040:	*	:00000.0- :00
1A8: -0.001877:	*	:00000.0- :00
1A9: -0.002035:	*	:00000.0- :00
1AA: -0.001921:	*	:00000.0- :00
1AB: -0.001896:	*	:00000.0- :00
1AC: -0.001867:	*	:00000.0- :00
1AD: -0.002108:	*	:00000.0- :00
1AE: -0.001997:	*	:00000.0- :00
1AF: -0.001807:	*	:00000.0- :00
1B0: -0.001822:	*	:00000.0- :00
1B1: -0.002051:	*	:00000.0- :00
1B2: -0.001916:	*	:00000.0- :00
1B3: -0.001991:	*	:00000.0- :00
1B4: -0.001973:	*	:00000.0- :00
1B5: -0.002108:	*	:00000.0- :00
1B6: -0.002067:	*	:00000.0- :00
1B7: -0.002013:	*	:00000.0- :00
1B8: -0.002053:	*	:00000.0- :00
1B9: -0.002113:	*	:00000.0- :00
1BA: -0.001913:	*	:00000.0- :00
1BB: -0.001950:	*	:00000.0- :00
1BC: -0.001974:	*	:00000.0- :00
1BD: -0.002144:	*	:00000.0- :00
1BE: -0.002055:	*	:00000.0- :00
1BF: -0.002041:	*	:00000.0- :00
1C0: -0.000629:	*	:00000.0- :00
1C1: -0.000747:	*	:00000.0- :00
1C2: -0.000569:	*	:00000.0- :00
1C3: -0.000701:	*	:00000.0- :00
1C4: -0.000497:	*	:00000.0- :00
1C5: -0.000746:	*	:00000.0- :00
1C6: -0.000533:	*	:00000.0- :00
1C7: -0.000677:	*	:00000.0- :00
1C8: -0.000419:	*	:00000.0- :00
1C9: -0.000728:	*	:00000.0- :00
1CA: -0.000699:	*	:00000.0- :00
1CB: -0.000643:	*	:00000.0- :00
1CC: -0.000509:	*	:00000.0- :00
1CD: -0.000759:	*	:00000.0- :00
1CE: -0.000676:	*	:00000.0- :00
1CF: -0.000678:	*	:00000.0- :00
1D0: -0.000508:	*	:00000.0- :00
1D1: -0.000778:	*	:00000.0- :00
1D2: -0.000675:	*	:00000.0- :00
1D3: -0.000726:	*	:00000.0- :00
1D4: -0.000558:	*	:00000.0- :00
1D5: -0.000768:	*	:00000.0- :00
1D6: -0.000760:	*	:00000.0- :00
1D7: -0.000645:	*	:00000.0- :00
1D8: -0.000580:	*	:00000.0- :00

1A-000000

270365-40

Non. Lin. Error, SN = 4130 (Continued)

1D9: -0.000836:	*	:248*0.0-:100
1DA: -0.000750:	*	:249*0.0-:100
1DB: -0.000758:	*	:250*0.0-:100
1DC: -0.000715:	*	:251*0.0-:100
1DD: -0.000816:	*	:252*0.0-:100
1DE: -0.000701:	*	:253*0.0-:100
1DF: -0.000714:	*	:254*0.0-:100
1E0: -0.000663:	*	:255*0.0-:100
1E1: -0.001060:	*	:256*0.0-:100
1E2: -0.000828:	*	:257*0.0-:100
1E3: -0.001107:	*	:258*0.0-:100
1E4: -0.000802:	*	:259*0.0-:100
1E5: -0.001037:	*	:260*0.0-:100
1E6: -0.000803:	*	:261*0.0-:100
1E7: -0.000843:	*	:262*0.0-:100
1E8: -0.000702:	*	:263*0.0-:100
1E9: -0.001156:	*	:264*0.0-:100
1EA: -0.000861:	*	:265*0.0-:100
1EB: -0.000965:	*	:266*0.0-:100
1EC: -0.000933:	*	:267*0.0-:100
1ED: -0.001142:	*	:268*0.0-:100
1EE: -0.001205:	*	:269*0.0-:100
1EF: -0.000995:	*	:270*0.0-:100
1F0: -0.000954:	*	:271*0.0-:100
1F1: -0.001179:	*	:272*0.0-:100
1F2: -0.001084:	*	:273*0.0-:100
1F3: -0.001061:	*	:274*0.0-:100
1F4: -0.001035:	*	:275*0.0-:100
1F5: -0.001099:	*	:276*0.0-:100
1F6: -0.001111:	*	:277*0.0-:100
1F7: -0.000960:	*	:278*0.0-:100
1F8: -0.000991:	*	:279*0.0-:100
1F9: -0.001178:	*	:280*0.0-:100
1FA: -0.001061:	*	:281*0.0-:100
1FB: -0.001099:	*	:282*0.0-:100
1FC: -0.001026:	*	:283*0.0-:100
1FD: -0.001248:	*	:284*0.0-:100
1FE: -0.001157:	*	:285*0.0-:100
1FF: -0.001828:	*	:286*0.0-:100
200: -0.001360:	*	:287*0.0-:100
201: -0.001583:	*	:288*0.0-:100
202: -0.001386:	*	:289*0.0-:100
203: -0.001636:	*	:290*0.0-:100
204: -0.001536:	*	:291*0.0-:100
205: -0.001584:	*	:292*0.0-:100
206: -0.001514:	*	:293*0.0-:100
207: -0.001703:	*	:294*0.0-:100
208: -0.001714:	*	:295*0.0-:100
209: -0.002021:	*	:296*0.0-:100
20A: -0.001592:	*	:297*0.0-:100
20B: -0.001799:	*	:298*0.0-:100
20C: -0.001884:	*	:299*0.0-:100
20D: -0.001994:	*	:300*0.0-:100
20E: -0.002028:	*	:301*0.0-:100
20F: -0.002225:	*	:302*0.0-:100
210: -0.001805:	*	:303*0.0-:100
211: -0.002137:	*	:304*0.0-:100
212: -0.001994:	*	:305*0.0-:100
213: -0.002071:	*	:306*0.0-:100
214: -0.001795:	*	:307*0.0-:100

0A-8000'S

270365-41

Non. Lin. Error, SN = 4130 (Continued)

215: -0.002104:	*	:C85100.0- :11
216: -0.001945:	*	:680100.0- :15
217: -0.002001:	*	:105100.0- :16
218: -0.001974:	*	:022000.0- :18
219: -0.002175:	*	:001100.0- :12
21A: -0.002187:	*	:PC8100.0- :10
21B: -0.002104:	*	:TA1100.0- :17
21C: -0.001725:	*	:020100.0- :18
21D: -0.002212:	*	:T11100.0- :19
21E: -0.002012:	*	:S0C100.0- :14
21F: -0.002564:	*	:+15100.0- :10
220: -0.002027:	*	:400100.0- :13
221: -0.002294:	*	:0C5100.0- :10
222: -0.002127:	*	:CY1100.0- :13
223: -0.002269:	*	:E31100.0- :19
224: -0.002157:	*	:P55100.0- :10
225: -0.002308:	*	:+02100.0- :11
226: -0.002268:	*	:VY1100.0- :15
227: -0.002372:	*	:051100.0- :10
228: -0.002039:	*	:01C100.0- :14
229: -0.002344:	*	:T13100.0- :10
22A: -0.002218:	*	:506100.0- :13
22B: -0.002244:	*	:006100.0- :19
22C: -0.002179:	*	:+05100.0- :10
22D: -0.002361:	*	:+03100.0- :10
22E: -0.002101:	*	:P0C100.0- :14
22F: -0.002463:	*	:022100.0- :10
230: -0.002088:	*	:+502100.0- :13
231: -0.002333:	*	:067100.0- :10
232: -0.002052:	*	:+527100.0- :13
233: -0.002328:	*	:+06100.0- :19
234: -0.002104:	*	:+02100.0- :10
235: -0.002278:	*	:087100.0- :11
236: -0.002357:	*	:+097100.0- :13
237: -0.002259:	*	:127100.0- :10
238: -0.002078:	*	:017100.0- :14
239: -0.002559:	*	:000100.0- :12
23A: -0.002199:	*	:+C10100.0- :10
23B: -0.002343:	*	:1010100.0- :17
23C: -0.002181:	*	:+C0100.0- :10
23D: -0.002369:	*	:+00000.0- :10
23E: -0.002265:	*	:000100.0- :14
23F: -0.002833:	*	:000500.0- :10
240: -0.001187:	*	:087100.0- :13
241: -0.001357:	*	:+005500.0- :10
242: -0.001130:	*	:+00100.0- :10
243: -0.001301:	*	:+101500.0- :17
244: -0.001167:	*	:+00100.0- :10
245: -0.001462:	*	:+012100.0- :14
246: -0.001157:	*	:+001100.0- :15
247: -0.001412:	*	:+007100.0- :10
248: -0.001224:	*	:+01C100.0- :14
249: -0.001278:	*	:+210700.0- :10
24A: -0.001187:	*	:+000100.0- :10
24B: -0.001278:	*	:+000100.0- :10
24C: -0.001023:	*	:+0100.0- :10
24D: -0.001256:	*	:+007100.0- :10
24E: -0.001147:	*	:+00100.0- :14
24F: -0.001204:	*	:+000100.0- :10
250: -0.000930:	*	:+001100.0- :17

2A-380015

270365-42

Non. Lin. Error, SN = 4130 (Continued)

251: -0.001283:	*	*001500.0- :00E
252: -0.001088:	*	*00100.0- :00E
253: -0.001281:	*	*000500.0- :00E
254: -0.000930:	*	*000000.0- :00E
255: -0.001188:	*	*001500.0- :00E
256: -0.001039:	*	*001500.0- :00E
257: -0.001147:	*	*001500.0- :00E
258: -0.001096:	*	*00100.0- :00E
259: -0.001217:	*	*001500.0- :00E
25A: -0.001302:	*	*000500.0- :00E
25B: -0.001214:	*	*000500.0- :00E
25C: -0.001084:	*	*000500.0- :00E
25D: -0.001276:	*	*000500.0- :00E
25E: -0.001273:	*	*001500.0- :00E
25F: -0.001343:	*	*000500.0- :00E
260: -0.001229:	*	*001500.0- :00E
261: -0.001509:	*	*000500.0- :00E
262: -0.001297:	*	*000500.0- :00E
263: -0.001426:	*	*000500.0- :00E
264: -0.001318:	*	*000500.0- :00E
265: -0.001517:	*	*000500.0- :00E
266: -0.001282:	*	*001500.0- :00E
267: -0.001485:	*	*000500.0- :00E
268: -0.001357:	*	*001500.0- :00E
269: -0.001616:	*	*000500.0- :00E
26A: -0.001509:	*	*001500.0- :00E
26B: -0.001558:	*	*000500.0- :00E
26C: -0.001582:	*	*000500.0- :00E
26D: -0.001748:	*	*000500.0- :00E
26E: -0.001524:	*	*000500.0- :00E
26F: -0.001692:	*	*000500.0- :00E
270: -0.001589:	*	*001500.0- :00E
271: -0.001786:	*	*001500.0- :00E
272: -0.001708:	*	*000500.0- :00E
273: -0.001751:	*	*000500.0- :00E
274: -0.001716:	*	*000500.0- :00E
275: -0.001988:	*	*000500.0- :00E
276: -0.001813:	*	*001500.0- :00E
277: -0.001816:	*	*000500.0- :00E
278: -0.001943:	*	*001500.0- :00E
279: -0.002046:	*	*000500.0- :00E
27A: -0.001936:	*	*000500.0- :00E
27B: -0.002000:	*	*000500.0- :00E
27C: -0.001783:	*	*00100.0- :00E
27D: -0.002248:	*	*000500.0- :00E
27E: -0.001869:	*	*00100.0- :00E
27F: -0.002131:	*	*00100.0- :00E
280: -0.001496:	*	*00100.0- :00E
281: -0.001510:	*	*00100.0- :00E
282: -0.001316:	*	*00100.0- :00E
283: -0.001792:	*	*00100.0- :00E
284: -0.001318:	*	*000500.0- :00E
285: -0.001615:	*	*000500.0- :00E
286: -0.001465:	*	*00100.0- :00E
287: -0.001632:	*	*000500.0- :00E
288: -0.001643:	*	*000500.0- :00E
289: -0.001730:	*	*000500.0- :00E
28A: -0.001629:	*	*00100.0- :00E
28B: -0.001698:	*	*000500.0- :00E
28C: -0.001823:	*	*000500.0- :00E

SP-286/VS2

270365-43

Non. Lin. Error, SN = 4130 (Continued)

28D: -0.001855:	*	:000100.0- :000
28E: -0.001778:	*	:000100.0- :000
28F: -0.001942:	*	:000100.0- :000
290: -0.001871:	*	:000100.0- :000
291: -0.002008:	*	:000100.0- :000
292: -0.001945:	*	:000100.0- :000
293: -0.002128:	*	:000100.0- :000
294: -0.001962:	*	:000100.0- :000
295: -0.002235:	*	:000100.0- :000
296: -0.002101:	*	:000100.0- :000
297: -0.002178:	*	:000100.0- :000
298: -0.002132:	*	:000100.0- :000
299: -0.002366:	*	:000100.0- :000
29A: -0.002433:	*	:000100.0- :000
29B: -0.002360:	*	:000100.0- :000
29C: -0.002236:	*	:000100.0- :000
29D: -0.002541:	*	:000100.0- :000
29E: -0.002403:	*	:000100.0- :000
29F: -0.002609:	*	:000100.0- :000
2A0: -0.002413:	*	:000100.0- :000
2A1: -0.002563:	*	:000100.0- :000
2A2: -0.002381:	*	:000100.0- :000
2A3: -0.002542:	*	:000100.0- :000
2A4: -0.002435:	*	:000100.0- :000
2A5: -0.002697:	*	:000100.0- :000
2A6: -0.002530:	*	:000100.0- :000
2A7: -0.002653:	*	:000100.0- :000
2A8: -0.002459:	*	:000100.0- :000
2A9: -0.002742:	*	:000100.0- :000
2AA: -0.002860:	*	:000100.0- :000
2AB: -0.002666:	*	:000100.0- :000
2AC: -0.002525:	*	:000100.0- :000
2AD: -0.002741:	*	:000100.0- :000
2AE: -0.002785:	*	:000100.0- :000
2AF: -0.002737:	*	:000100.0- :000
2B0: -0.002709:	*	:000100.0- :000
2B1: -0.003031:	*	:000100.0- :000
2B2: -0.002823:	*	:000100.0- :000
2B3: -0.002906:	*	:000100.0- :000
2B4: -0.002780:	*	:000100.0- :000
2B5: -0.003019:	*	:000100.0- :000
2B6: -0.002941:	*	:000100.0- :000
2B7: -0.002923:	*	:000100.0- :000
2B8: -0.002794:	*	:000100.0- :000
2B9: -0.002973:	*	:000100.0- :000
2BA: -0.002872:	*	:000100.0- :000
2BB: -0.002943:	*	:000100.0- :000
2BC: -0.002584:	*	:000100.0- :000
2BD: -0.002832:	*	:000100.0- :000
2BE: -0.002777:	*	:000100.0- :000
2BF: -0.002698:	*	:000100.0- :000
2C0: -0.001295:	*	:000100.0- :000
2C1: -0.001557:	*	:000100.0- :000
2C2: -0.001429:	*	:000100.0- :000
2C3: -0.001542:	*	:000100.0- :000
2C4: -0.001274:	*	:000100.0- :000
2C5: -0.001417:	*	:000100.0- :000
2C6: -0.001409:	*	:000100.0- :000
2C7: -0.001382:	*	:000100.0- :000
2C8: -0.001138:	*	:000100.0- :000

22-286075

270365-44

Non. Lin. Error, SN = 4130 (Continued)

2C9: -0.001450:	*	†	228100.0- 1015
2CA: -0.001409:	*	†	227100.0- 1015
2CB: -0.001366:	*	†	226100.0- 1015
2CC: -0.001201:	*	†	225100.0- 1015
2CD: -0.001385:	*	†	224100.0- 1015
2CE: -0.001406:	*	†	223100.0- 1015
2CF: -0.001532:	*	†	222100.0- 1015
2D0: -0.001166:	*	†	221100.0- 1015
2D1: -0.001503:	*	†	220100.0- 1015
2D2: -0.001411:	*	†	219100.0- 1015
2D3: -0.001554:	*	†	218100.0- 1015
2D4: -0.001334:	*	†	217100.0- 1015
2D5: -0.001622:	*	†	216100.0- 1015
2D6: -0.001370:	*	†	215100.0- 1015
2D7: -0.001369:	*	†	214100.0- 1015
2D8: -0.001438:	*	†	213100.0- 1015
2D9: -0.001660:	*	†	212100.0- 1015
2DA: -0.001324:	*	†	211100.0- 1015
2DB: -0.001395:	*	†	210100.0- 1015
2DC: -0.001273:	*	†	209100.0- 1015
2DD: -0.001813:	*	†	208100.0- 1015
2DE: -0.001428:	*	†	207100.0- 1015
2DF: -0.001550:	*	†	206100.0- 1015
2E0: -0.001685:	*	†	205100.0- 1015
2E1: -0.001799:	*	†	204100.0- 1015
2E2: -0.001725:	*	†	203100.0- 1015
2E3: -0.001766:	*	†	202100.0- 1015
2E4: -0.001954:	*	†	201100.0- 1015
2E5: -0.001920:	*	†	200100.0- 1015
2E6: -0.001747:	*	†	199100.0- 1015
2E7: -0.001796:	*	†	198100.0- 1015
2E8: -0.001776:	*	†	197100.0- 1015
2E9: -0.002011:	*	†	196100.0- 1015
2EA: -0.001834:	*	†	195100.0- 1015
2EB: -0.002115:	*	†	194100.0- 1015
2EC: -0.001915:	*	†	193100.0- 1015
2ED: -0.002089:	*	†	192100.0- 1015
2EE: -0.002046:	*	†	191100.0- 1015
2EF: -0.002132:	*	†	190100.0- 1015
2F0: -0.002069:	*	†	189100.0- 1015
2F1: -0.002229:	*	†	188100.0- 1015
2F2: -0.002101:	*	†	187100.0- 1015
2F3: -0.002236:	*	†	186100.0- 1015
2F4: -0.002177:	*	†	185100.0- 1015
2F5: -0.002388:	*	†	184100.0- 1015
2F6: -0.002284:	*	†	183100.0- 1015
2F7: -0.002315:	*	†	182100.0- 1015
2F8: -0.002449:	*	†	181100.0- 1015
2F9: -0.002533:	*	†	180100.0- 1015
2FA: -0.002538:	*	†	179100.0- 1015
2FB: -0.002564:	*	†	178100.0- 1015
2FC: -0.002471:	*	†	177100.0- 1015
2FD: -0.002486:	*	†	176100.0- 1015
2FE: -0.002576:	*	†	175100.0- 1015
2FF: -0.002006:	*	†	174100.0- 1015
300: -0.001994:	*	†	173100.0- 1015
301: -0.002301:	*	†	172100.0- 1015
302: -0.002168:	*	†	171100.0- 1015
303: -0.002284:	*	†	170100.0- 1015
304: -0.002251:	*	†	169100.0- 1015

66-28073

270365-45

Non. Lin. Error, SN = 4130 (Continued)

305: -0.002417:	*	:000100.0- :134
306: -0.002269:	*	:000000.0- :134
307: -0.002483:	*	:001100.0- :134
308: -0.002265:	*	:002000.0- :134
309: -0.002589:	*	:003000.0- :134
30A: -0.002383:	*	:004000.0- :134
30B: -0.002508:	*	:005000.0- :134
30C: -0.002336:	*	:006000.0- :134
30D: -0.002560:	*	:007000.0- :134
30E: -0.002428:	*	:008000.0- :134
30F: -0.002677:	*	:009000.0- :134
310: -0.002528:	*	:010000.0- :134
311: -0.002861:	*	:011000.0- :134
312: -0.002612:	*	:012000.0- :134
313: -0.002746:	*	:013000.0- :134
314: -0.002710:	*	:014000.0- :134
315: -0.002955:	*	:015000.0- :134
316: -0.002813:	*	:016000.0- :134
317: -0.002864:	*	:017000.0- :134
318: -0.002770:	*	:018000.0- :134
319: -0.002959:	*	:019000.0- :134
31A: -0.002888:	*	:020000.0- :134
31B: -0.002901:	*	:021000.0- :134
31C: -0.002742:	*	:022000.0- :134
31D: -0.002975:	*	:023000.0- :134
31E: -0.002878:	*	:024000.0- :134
31F: -0.003165:	*	:025000.0- :134
320: -0.002991:	*	:026000.0- :134
321: -0.003220:	*	:027000.0- :134
322: -0.003083:	*	:028000.0- :134
323: -0.003195:	*	:029000.0- :134
324: -0.003109:	*	:030000.0- :134
325: -0.003314:	*	:031000.0- :134
326: -0.003130:	*	:032000.0- :134
327: -0.003246:	*	:033000.0- :134
328: -0.003301:	*	:034000.0- :134
329: -0.003397:	*	:035000.0- :134
32A: -0.003247:	*	:036000.0- :134
32B: -0.003362:	*	:037000.0- :134
32C: -0.002182:	*	:038000.0- :134
32D: -0.002338:	*	:039000.0- :134
32E: -0.002251:	*	:040000.0- :134
32F: -0.002332:	*	:041000.0- :134
330: -0.001979:	*	:042000.0- :134
331: -0.002225:	*	:043000.0- :134
332: -0.002099:	*	:044000.0- :134
333: -0.002164:	*	:045000.0- :134
334: -0.001894:	*	:046000.0- :134
335: -0.002134:	*	:047000.0- :134
336: -0.002018:	*	:048000.0- :134
337: -0.002019:	*	:049000.0- :134
338: -0.001991:	*	:050000.0- :134
339: -0.002182:	*	:051000.0- :134
33A: -0.002183:	*	:052000.0- :134
33B: -0.002134:	*	:053000.0- :134
33C: -0.002005:	*	:054000.0- :134
33D: -0.002338:	*	:055000.0- :134
33E: -0.002115:	*	:056000.0- :134
33F: -0.002380:	*	:057000.0- :134
340: -0.000653:	*	:058000.0- :134

71-200073

270365-46

Non. Lin. Error, SN = 4130 (Continued)

341: -0.001006:	*	172500.0-180
342: -0.000888:	*	173500.0-180
343: -0.001175:	*	174500.0-180
344: -0.000834:	*	175500.0-180
345: -0.000951:	*	176500.0-180
346: -0.001030:	*	177500.0-180
347: -0.000951:	*	178500.0-180
348: -0.000710:	*	179500.0-180
349: -0.000672:	*	180500.0-180
34A: -0.000854:	*	181500.0-180
34B: -0.000934:	*	182500.0-180
34C: -0.000648:	*	183500.0-180
34D: -0.001008:	*	184500.0-180
34E: -0.000828:	*	185500.0-180
34F: -0.000777:	*	186500.0-180
350: -0.000774:	*	187500.0-180
351: -0.000994:	*	188500.0-180
352: -0.000901:	*	189500.0-180
353: -0.000985:	*	190500.0-180
354: -0.000618:	*	191500.0-180
355: -0.000920:	*	192500.0-180
356: -0.000731:	*	193500.0-180
357: -0.000707:	*	194500.0-180
358: -0.000832:	*	195500.0-180
359: -0.001047:	*	196500.0-180
35A: -0.001003:	*	197500.0-180
35B: -0.000976:	*	198500.0-180
35C: -0.000957:	*	199500.0-180
35D: -0.001273:	*	200500.0-180
35E: -0.000994:	*	201500.0-180
35F: -0.001038:	*	202500.0-180
360: -0.001150:	*	203500.0-180
361: -0.001320:	*	204500.0-180
362: -0.001257:	*	205500.0-180
363: -0.001390:	*	206500.0-180
364: -0.001263:	*	207500.0-180
365: -0.001498:	*	208500.0-180
366: -0.001388:	*	209500.0-180
367: -0.001453:	*	210500.0-180
368: -0.001295:	*	211500.0-180
369: -0.001416:	*	212500.0-180
36A: -0.001389:	*	213500.0-180
36B: -0.001498:	*	214500.0-180
36C: -0.001502:	*	215500.0-180
36D: -0.001797:	*	216500.0-180
36E: -0.001566:	*	217500.0-180
36F: -0.001596:	*	218500.0-180
370: -0.001531:	*	219500.0-180
371: -0.001799:	*	220500.0-180
372: -0.001684:	*	221500.0-180
373: -0.001735:	*	222500.0-180
374: -0.001647:	*	223500.0-180
375: -0.001857:	*	224500.0-180
376: -0.001809:	*	225500.0-180
377: -0.001697:	*	226500.0-180
378: -0.001770:	*	227500.0-180
379: -0.001956:	*	228500.0-180
37A: -0.001821:	*	229500.0-180
37B: -0.001841:	*	230500.0-180
37C: -0.001820:	*	231500.0-180

84-28809

270365-47

Non. Lin. Error, SN = 4130 (Continued)

37D: -0.001979:	*	116000.0-1000
37E: -0.002106:	*	116000.0-1000
37F: -0.001597:	*	116000.0-1000
380: -0.001784:	*	116000.0-1000
381: -0.002085:	*	116000.0-1000
382: -0.001840:	*	116000.0-1000
383: -0.001907:	*	116000.0-1000
384: -0.001890:	*	116000.0-1000
385: -0.001989:	*	116000.0-1000
386: -0.001867:	*	116000.0-1000
387: -0.001957:	*	116000.0-1000
388: -0.002029:	*	116000.0-1000
389: -0.002256:	*	116000.0-1000
38A: -0.002177:	*	116000.0-1000
38B: -0.002285:	*	116000.0-1000
38C: -0.002294:	*	116000.0-1000
38D: -0.002497:	*	116000.0-1000
38E: -0.002204:	*	116000.0-1000
38F: -0.002499:	*	116000.0-1000
390: -0.002201:	*	116000.0-1000
391: -0.002774:	*	116000.0-1000
392: -0.002398:	*	116000.0-1000
393: -0.002591:	*	116000.0-1000
394: -0.002325:	*	116000.0-1000
395: -0.002570:	*	116000.0-1000
396: -0.002590:	*	116000.0-1000
397: -0.002513:	*	116000.0-1000
398: -0.002409:	*	116000.0-1000
399: -0.002662:	*	116000.0-1000
39A: -0.002513:	*	116000.0-1000
39B: -0.002588:	*	116000.0-1000
39C: -0.002345:	*	116000.0-1000
39D: -0.002633:	*	116000.0-1000
39E: -0.002485:	*	116000.0-1000
39F: -0.002579:	*	116000.0-1000
3A0: -0.002427:	*	116000.0-1000
3A1: -0.002646:	*	116000.0-1000
3A2: -0.002572:	*	116000.0-1000
3A3: -0.002639:	*	116000.0-1000
3A4: -0.002393:	*	116000.0-1000
3A5: -0.002494:	*	116000.0-1000
3A6: -0.002609:	*	116000.0-1000
3A7: -0.002570:	*	116000.0-1000
3A8: -0.002522:	*	116000.0-1000
3A9: -0.002809:	*	116000.0-1000
3AA: -0.002658:	*	116000.0-1000
3AB: -0.002698:	*	116000.0-1000
3AC: -0.002645:	*	116000.0-1000
3AD: -0.002724:	*	116000.0-1000
3AE: -0.002721:	*	116000.0-1000
3AF: -0.002685:	*	116000.0-1000
3B0: -0.002752:	*	116000.0-1000
3B1: -0.003015:	*	116000.0-1000
3B2: -0.002837:	*	116000.0-1000
3B3: -0.002932:	*	116000.0-1000
3B4: -0.002689:	*	116000.0-1000
3B5: -0.003034:	*	116000.0-1000
3B6: -0.002817:	*	116000.0-1000
3B7: -0.002812:	*	116000.0-1000
3B8: -0.002965:	*	116000.0-1000

06-2507

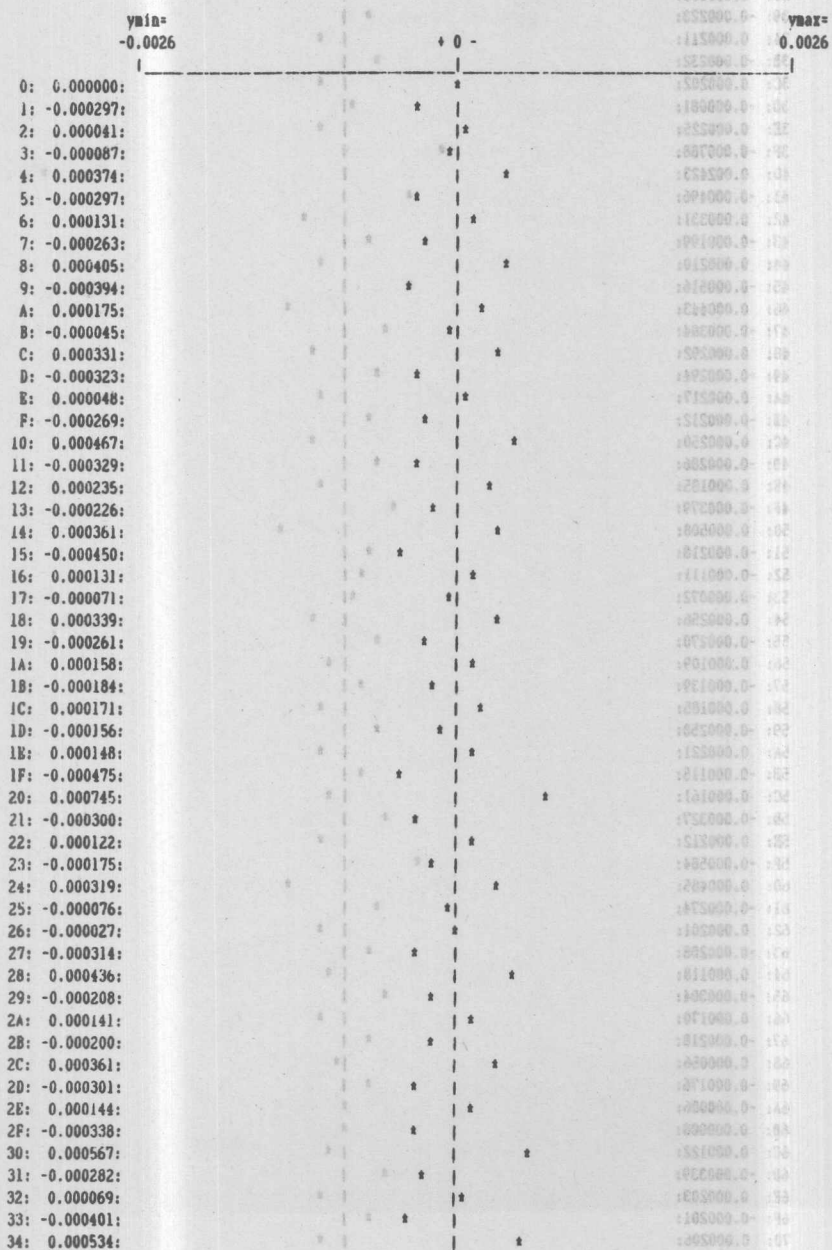
06-2507

270365-48

Non. Lin. Error, SN = 4130 (Continued)

270365-50

DNL Error, SN = 4130



68-330012

270365-87

DNL Error, SN = 4130

270365-88

6-262

71: -0.000356:	*		005000.0 -18
72: 0.000133:			006000.0 -19
73: -0.000038:			007000.0 -20
74: 0.000078:			008000.0 -21
75: -0.000400:	*		009000.0 -22
76: 0.000277:			010000.0 -23
77: -0.000100:			011000.0 -24
78: -0.000063:			012000.0 -25
79: -0.000218:	*		013000.0 -26
7A: 0.000104:			014000.0 -27
7B: -0.000058:			015000.0 -28
7C: -0.000009:			016000.0 -29
7D: -0.000209:	*		017000.0 -30
7E: 0.000141:			018000.0 -31
7F: -0.000683:	*		019000.0 -32
80: 0.001293:			020000.0 -33
81: -0.000244:	*		021000.0 -34
82: 0.000141:			022000.0 -35
83: -0.000154:			023000.0 -36
84: 0.000131:			024000.0 -37
85: -0.000300:	*		025000.0 -38
86: 0.000131:			026000.0 -39
87: -0.000240:	*		027000.0 -40
88: 0.000239:			028000.0 -41
89: -0.000593:	*		029000.0 -42
8A: 0.000535:			030000.0 -43
8B: -0.000361:	*		031000.0 -44
8C: 0.000188:			032000.0 -45
8D: -0.000271:	*		033000.0 -46
8E: -0.000059:			034000.0 -47
8F: -0.000121:	*		035000.0 -48
90: 0.000374:			036000.0 -49
91: -0.000145:	*		037000.0 -50
92: 0.000113:			038000.0 -51
93: -0.000187:	*		039000.0 -52
94: 0.000154:			040000.0 -53
95: -0.000219:	*		041000.0 -54
96: 0.000145:			042000.0 -55
97: -0.000207:	*		043000.0 -56
98: 0.000136:			044000.0 -57
99: -0.000326:	*		045000.0 -58
9A: 0.000230:			046000.0 -59
9B: -0.000161:	*		047000.0 -60
9C: 0.000070:			048000.0 -61
9D: -0.000345:	*		049000.0 -62
9E: 0.000183:			050000.0 -63
9F: -0.000500:	*		051000.0 -64
A0: 0.000485:			052000.0 -65
A1: -0.000349:	*		053000.0 -66
A2: 0.000414:			054000.0 -67
A3: -0.000374:	*		055000.0 -68
A4: 0.000215:			056000.0 -69
A5: -0.000273:	*		057000.0 -70
A6: 0.000138:			058000.0 -71
A7: -0.000189:	*		059000.0 -72
A8: 0.000124:			060000.0 -73
A9: -0.000118:	*		061000.0 -74
AA: 0.000123:			062000.0 -75
AB: -0.000142:	*		063000.0 -76
AC: 0.000086:			064000.0 -77

09-288075

270365-89

DNL Error, SN = 4130 (Continued)

AD: -0.000273: * | * :02000.0 :1F
AE: 0.000137: * | * :02100.0 :5F
AF: -0.000194: * | * :02200.0 :9F
B0: 0.000366: * | * :02300.0 :DF
B1: -0.000233: * | * :02400.0 :1F
B2: 0.000111: * | * :02500.0 :5F
B3: -0.000115: * | * :02600.0 :9F
B4: -0.000039: * | * :02700.0 :DF
B5: -0.000088: * | * :02800.0 :1F
B6: 0.000100: * | * :02900.0 :5F
B7: -0.000147: * | * :03000.0 :9F
B8: 0.000109: * | * :03100.0 :DF
B9: -0.000171: * | * :03200.0 :1F
BA: 0.000156: * | * :03300.0 :5F
BB: -0.000180: * | * :03400.0 :9F
BC: 0.000041: * | * :03500.0 :DF
BD: -0.000134: * | * :03600.0 :1F
BE: 0.000202: * | * :03700.0 :5F
BF: -0.000561: * | * :03800.0 :9F
C0: 0.002134: * | * :03900.0 :DF
C1: -0.000301: * | * :04000.0 :1F
C2: 0.000150: * | * :04100.0 :5F
C3: -0.000138: * | * :04200.0 :9F
C4: 0.000206: * | * :04300.0 :DF
C5: -0.000371: * | * :04400.0 :1F
C6: 0.000377: * | * :04500.0 :5F
C7: -0.000341: * | * :04600.0 :9F
C8: 0.000272: * | * :04700.0 :DF
C9: -0.000235: * | * :04800.0 :1F
CA: -0.000105: * | * :04900.0 :5F
CB: 0.000091: * | * :05000.0 :9F
CC: 0.000203: * | * :05100.0 :DF
CD: -0.000322: * | * :05200.0 :1F
CE: 0.000207: * | * :05300.0 :5F
CF: -0.000187: * | * :05400.0 :9F
D0: 0.000151: * | * :05500.0 :DF
D1: -0.000250: * | * :05600.0 :1F
D2: 0.000142: * | * :05700.0 :5F
D3: -0.000396: * | * :05800.0 :9F
D4: 0.000501: * | * :05900.0 :DF
D5: -0.000362: * | * :06000.0 :1F
D6: 0.000329: * | * :06100.0 :5F
D7: -0.000169: * | * :06200.0 :9F
D8: 0.000029: * | * :06300.0 :DF
D9: -0.000115: * | * :06400.0 :1F
DA: 0.000186: * | * :06500.0 :5F
DB: -0.000113: * | * :06600.0 :9F
DC: -0.000010: * | * :06700.0 :DF
DD: -0.000287: * | * :06800.0 :1F
DE: 0.000153: * | * :06900.0 :5F
DF: -0.000331: * | * :07000.0 :9F
E0: 0.000308: * | * :07100.0 :DF
E1: -0.000506: * | * :07200.0 :1F
E2: 0.000409: * | * :07300.0 :5F
E3: -0.000184: * | * :07400.0 :9F
E4: -0.000124: * | * :07500.0 :DF
E5: 0.000020: * | * :07600.0 :1F
E6: 0.000181: * | * :07700.0 :5F
E7: -0.000145: * | * :07800.0 :9F
E8: 0.000210: * | * :07900.0 :DF

GB-380072

270365-90

DNL Error, SN = 4130 (Continued)

E9: -0.000232:	*		:090000.0- :051
EA: 0.000136:	*		:091000.0- :051
EB: -0.000151:	*		:092000.0- :051
EC: 0.000168:	*		:093000.0- :051
ED: -0.000212:	*		:094000.0- :051
EE: 0.000195:	*		:095000.0- :051
EF: -0.000171:	*		:096000.0- :051
FO: 0.000115:	*		:097000.0- :051
F1: -0.000376:	*		:098000.0- :051
F2: 0.000208:	*		:099000.0- :051
F3: -0.000167:	*		:100000.0- :051
F4: 0.000078:	*		:101000.0- :051
F5: -0.000287:	*		:102000.0- :051
F6: 0.000348:	*		:103000.0- :051
F7: -0.000231:	*		:104000.0- :051
F8: -0.000161:	*		:105000.0- :051
F9: -0.000022:	*		:106000.0- :051
FA: 0.000265:	*		:107000.0- :051
FB: -0.000130:	*		:108000.0- :051
FC: 0.000167:	*		:109000.0- :051
FD: -0.000144:	*		:110000.0- :051
FE: 0.000086:	*		:111000.0- :051
FF: -0.000880:	*		:112000.0- :051
100: 0.001348:	*		:113000.0- :051
101: -0.000334:	*		:114000.0- :051
102: 0.000236:	*		:115000.0- :051
103: -0.000222:	*		:116000.0- :051
104: -0.000030:	*		:117000.0- :051
105: -0.000123:	*		:118000.0- :051
106: 0.000008:	*		:119000.0- :051
107: -0.000068:	*		:120000.0- :051
108: 0.000266:	*		:121000.0- :051
109: -0.000265:	*		:122000.0- :051
10A: 0.000136:	*		:123000.0- :051
10B: -0.000078:	*		:124000.0- :051
10C: 0.000069:	*		:125000.0- :051
10D: -0.000260:	*		:126000.0- :051
10E: 0.000142:	*		:127000.0- :051
10F: -0.000352:	*		:128000.0- :051
110: 0.000142:	*		:129000.0- :051
111: -0.000146:	*		:130000.0- :051
112: 0.000292:	*		:131000.0- :051
113: -0.000150:	*		:132000.0- :051
114: 0.000003:	*		:133000.0- :051
115: -0.000263:	*		:134000.0- :051
116: 0.000174:	*		:135000.0- :051
117: -0.000151:	*		:136000.0- :051
118: 0.000078:	*		:137000.0- :051
119: -0.000120:	*		:138000.0- :051
11A: 0.000027:	*		:139000.0- :051
11B: -0.000120:	*		:140000.0- :051
11C: 0.000048:	*		:141000.0- :051
11D: -0.000248:	*		:142000.0- :051
11E: 0.000192:	*		:143000.0- :051
11F: -0.000455:	*		:144000.0- :051
120: 0.000535:	*		:145000.0- :051
121: -0.000300:	*		:146000.0- :051
122: 0.000139:	*		:147000.0- :051
123: -0.000088:	*		:148000.0- :051
124: 0.000145:	*		:149000.0- :051

SE-88003

270365-91

DNL Error, SN = 4130 (Continued)

125: -0.000496:						125000.0- 125
126: 0.000193:						126000.0- 126
127: -0.000028:						127000.0- 127
128: 0.000202:						128000.0- 128
129: -0.000194:						129000.0- 129
12A: 0.000192:						130000.0- 130
12B: -0.000114:						131000.0- 131
12C: -0.000060:						132000.0- 132
12D: -0.000148:						133000.0- 133
12E: 0.000048:						134000.0- 134
12F: -0.000100:						135000.0- 135
130: 0.000168:						136000.0- 136
131: -0.000263:						137000.0- 137
132: 0.000188:						138000.0- 138
133: -0.000178:						139000.0- 139
134: 0.000193:						140000.0- 140
135: -0.000303:						141000.0- 141
136: 0.000259:						142000.0- 142
137: -0.000250:						143000.0- 143
138: 0.000436:						144000.0- 144
139: -0.000228:						145000.0- 145
13A: -0.000052:						146000.0- 146
13B: 0.000008:						147000.0- 147
13C: 0.000271:						148000.0- 148
13D: -0.000245:						149000.0- 149
13E: 0.000123:						150000.0- 150
13F: -0.000471:						151000.0- 151
140: 0.001823:						152000.0- 152
141: -0.000033:						153000.0- 153
142: 0.000139:						154000.0- 154
143: -0.000132:						155000.0- 155
144: 0.000199:						156000.0- 156
145: -0.000231:						157000.0- 157
146: 0.000116:						158000.0- 158
147: -0.000051:						159000.0- 159
148: 0.000217:						160000.0- 160
149: -0.000271:						161000.0- 161
14A: 0.000134:						162000.0- 162
14B: 0.000060:						163000.0- 163
14C: -0.000010:						164000.0- 164
14D: -0.000225:						165000.0- 165
14E: 0.000102:						166000.0- 166
14F: -0.000207:						167000.0- 167
150: 0.000168:						168000.0- 168
151: -0.000191:						169000.0- 169
152: 0.000133:						170000.0- 170
153: -0.000171:						171000.0- 171
154: 0.000170:						172000.0- 172
155: -0.000239:						173000.0- 173
156: 0.000056:						174000.0- 174
157: -0.000001:						175000.0- 175
158: 0.000199:						176000.0- 176
159: -0.000277:						177000.0- 177
15A: 0.000110:						178000.0- 178
15B: -0.000074:						179000.0- 179
15C: 0.000104:						180000.0- 180
15D: -0.000207:						181000.0- 181
15E: 0.000160:						182000.0- 182
15F: -0.000226:						183000.0- 183
160: 0.000138:						184000.0- 184

70-28075

270365-92

DNL Error, SN = 4130 (Continued)

161: -0.000271:	*	151000.0-:1001
162: 0.000215:		152000.0-:1021
163: -0.000217:	*	153000.0-:1031
164: 0.000226:		154000.0-:1041
165: -0.000454:	*	155000.0-:1051
166: 0.000347:		156100.0-:1061
167: -0.000119:	*	157000.0-:1071
168: 0.000173:		158000.0-:1081
169: -0.000158:	*	159000.0-:1091
16A: 0.000290:		160000.0-:1101
16B: -0.000237:	*	161000.0-:1111
16C: 0.000035:		162000.0-:1121
16D: -0.000279:	*	163000.0-:1131
16E: 0.000073:		164100.0-:1141
16F: 0.000027:	*	165000.0-:1151
170: 0.000069:		166000.0-:1161
171: -0.000279:	*	167000.0-:1171
172: 0.000191:		168100.0-:1181
173: -0.000064:	*	169000.0-:1191
174: 0.000101:		170000.0-:1201
175: -0.000192:	*	171000.0-:1211
176: 0.000065:		172000.0-:1221
177: -0.000025:	*	173000.0-:1231
178: 0.000164:		174000.0-:1241
179: -0.000301:	*	175000.0-:1251
17A: 0.000278:		176000.0-:1261
17B: -0.000146:	*	177000.0-:1271
17C: -0.000001:		178000.0-:1281
17D: -0.000193:	*	179000.0-:1291
17E: 0.000178:		180000.0-:1301
17F: -0.000110:	*	181000.0-:1311
180: 0.000512:		182000.0-:1321
181: -0.000281:	*	183000.0-:1331
182: 0.000073:		184000.0-:1341
183: -0.000060:	*	185000.0-:1351
184: 0.000135:		186100.0-:1361
185: -0.000247:	*	187000.0-:1371
186: 0.000100:		188000.0-:1381
187: -0.000103:	*	189000.0-:1391
188: 0.000171:		190000.0-:1401
189: -0.000244:	*	191000.0-:1411
18A: 0.000174:		192000.0-:1421
18B: -0.000114:	*	193000.0-:1431
18C: 0.000139:		194000.0-:1441
18D: -0.000329:	*	195000.0-:1451
18E: 0.000180:		196000.0-:1461
18F: -0.000176:	*	197000.0-:1471
190: 0.000448:		198000.0-:1481
191: -0.000395:	*	199000.0-:1491
192: -0.000046:		200000.0-:1501
193: -0.000003:	*	201000.0-:1511
194: 0.000001:		202000.0-:1521
195: -0.000168:	*	203000.0-:1531
196: 0.000171:		204000.0-:1541
197: -0.000355:	*	205000.0-:1551
198: 0.000558:		206000.0-:1561
199: -0.000405:	*	207000.0-:1571
19A: 0.000182:		208000.0-:1581
19B: -0.000093:	*	209000.0-:1591
19C: 0.000265:		210000.0-:1601

40-280012

270365-93

DNL Error, SN = 4130 (Continued)

19D: -0.000329:	* 175000.0: :131
19E: -0.000018:	* 175000.0: :501
19F: -0.000045:	* 175000.0: :C01
1A0: 0.000210:	* 175000.0: :101
1A1: -0.000244:	* 175000.0: :101
1A2: 0.000139:	* 175000.0: :101
1A3: -0.000091:	* 175000.0: :101
1A4: -0.000041:	* 175000.0: :101
1A5: -0.000210:	* 175000.0: :101
1A6: 0.000162:	* 175000.0: :101
1A7: -0.000057:	* 175000.0: :101
1A8: 0.000163:	* 175000.0: :101
1A9: -0.000158:	* 175000.0: :101
1AA: 0.000114:	* 175000.0: :101
1AB: 0.000024:	* 175000.0: :101
1AC: 0.000028:	* 175000.0: :101
1AD: -0.000240:	* 175000.0: :101
1AE: 0.000110:	* 175000.0: :101
1AF: 0.000189:	* 175000.0: :101
1B0: -0.000014:	* 175000.0: :101
1B1: -0.000229:	* 175000.0: :101
1B2: 0.000134:	* 175000.0: :101
1B3: -0.000075:	* 175000.0: :101
1B4: 0.000018:	* 175000.0: :101
1B5: -0.000135:	* 175000.0: :101
1B6: 0.000041:	* 175000.0: :101
1B7: 0.000053:	* 175000.0: :101
1B8: -0.000040:	* 175000.0: :101
1B9: -0.000060:	* 175000.0: :101
1BA: 0.000200:	* 175000.0: :101
1BB: -0.000037:	* 175000.0: :101
1BC: -0.000024:	* 175000.0: :101
1BD: -0.000169:	* 175000.0: :101
1BE: 0.000088:	* 175000.0: :101
1BF: 0.000013:	* 175000.0: :101
1C0: 0.001412:	* 175000.0: :101
1C1: -0.000118:	* 175000.0: :101
1C2: 0.000178:	* 175000.0: :101
1C3: -0.000132:	* 175000.0: :101
1C4: 0.000203:	* 175000.0: :101
1C5: -0.000248:	* 175000.0: :101
1C6: 0.000212:	* 175000.0: :101
1C7: -0.000144:	* 175000.0: :101
1C8: 0.000258:	* 175000.0: :101
1C9: -0.000309:	* 175000.0: :101
1CA: 0.000028:	* 175000.0: :101
1CB: 0.000056:	* 175000.0: :101
1CC: 0.000133:	* 175000.0: :101
1CD: -0.000250:	* 175000.0: :101
1CE: 0.000083:	* 175000.0: :101
1CF: -0.000002:	* 175000.0: :101
1D0: 0.000169:	* 175000.0: :101
1D1: -0.000269:	* 175000.0: :101
1D2: 0.000102:	* 175000.0: :101
1D3: -0.000051:	* 175000.0: :101
1D4: 0.000168:	* 175000.0: :101
1D5: -0.000210:	* 175000.0: :101
1D6: 0.000007:	* 175000.0: :101
1D7: 0.000115:	* 175000.0: :101
1D8: 0.000064:	* 175000.0: :101

270365-94

270365-94

DNL Error, SN = 4130 (Continued)

1D9: -0.000256:	*	000000.0 :015
1DA: 0.0000086:		001000.0 :015
1DB: -0.000008:	*	002000.0 :015
1DC: 0.000042:		003000.0 :015
1DD: -0.000101:	*	004000.0 :015
1DE: 0.000115:		005000.0 :015
1DF: -0.000013:	*	006000.0 :015
1E0: 0.000050:		007000.0 :015
1E1: -0.000396:	*	008000.0 :015
1E2: 0.000231:		009000.0 :015
1E3: -0.000279:	*	010000.0 :015
1E4: 0.000305:		011000.0 :015
1E5: -0.000235:	*	012000.0 :015
1E6: 0.000233:		013000.0 :015
1E7: -0.000039:	*	014000.0 :015
1E8: 0.000140:		015000.0 :015
1E9: -0.000454:	*	016000.0 :015
1EA: 0.000295:		017000.0 :015
1EB: -0.000104:	*	018000.0 :015
1EC: 0.000031:		019000.0 :015
1ED: -0.000208:	*	020000.0 :015
1EE: -0.000063:		021000.0 :015
1EF: 0.000209:	*	022000.0 :015
1F0: 0.000041:		023000.0 :015
1F1: -0.000225:	*	024000.0 :015
1F2: 0.000094:		025000.0 :015
1F3: 0.000023:	*	026000.0 :015
1F4: 0.000025:		027000.0 :015
1F5: -0.000064:	*	028000.0 :015
1F6: -0.000011:		029000.0 :015
1F7: 0.000150:	*	030000.0 :015
1F8: -0.000031:		031000.0 :015
1F9: -0.000186:	*	032000.0 :015
1FA: 0.000116:		033000.0 :015
1FB: -0.000038:	*	034000.0 :015
1FC: 0.000073:		035000.0 :015
1FD: -0.000222:	*	036000.0 :015
1FE: 0.000090:		037000.0 :015
1FF: -0.000671:	*	038000.0 :015
200: 0.000468:		039000.0 :015
201: -0.000223:	*	040000.0 :015
202: 0.000196:		041000.0 :015
203: -0.000249:	*	042000.0 :015
204: 0.000099:		043000.0 :015
205: -0.000048:	*	044000.0 :015
206: 0.000070:		045000.0 :015
207: -0.000189:	*	046000.0 :015
208: -0.000011:		047000.0 :015
209: -0.000307:	*	048000.0 :015
20A: 0.000429:		049000.0 :015
20B: -0.000207:	*	050000.0 :015
20C: -0.000085:		051000.0 :015
20D: -0.000109:	*	052000.0 :015
20E: -0.000034:		053000.0 :015
20F: -0.000197:	*	054000.0 :015
210: 0.000420:		055000.0 :015
211: -0.000332:	*	056000.0 :015
212: 0.000142:		057000.0 :015
213: -0.000076:	*	058000.0 :015
214: 0.000275:		059000.0 :015

88-862015

270365-95

DNL Error, SN = 4130 (Continued)

215: -0.000309:	*	:025000.0- :001
216: 0.000159:		:000000.0- :001
217: -0.000056:	*	:000000.0- :001
218: 0.000026:	*	:000000.0- :001
219: -0.000200:	*	:001000.0- :001
21A: -0.000012:	*	:011000.0- :001
21B: 0.000082:		:010000.0- :001
21C: 0.000379:		:020000.0- :001
21D: -0.000487:	*	:000000.0- :001
21E: 0.000199:		:000000.0- :001
21F: -0.000551:	*	:000000.0- :001
220: 0.000536:		:000000.0- :001
221: -0.000267:	*	:000000.0- :001
222: 0.000167:		:000000.0- :001
223: -0.000142:	*	:000000.0- :001
224: 0.000111:		:000000.0- :001
225: -0.000151:	*	:000000.0- :001
226: 0.000040:	*	:000000.0- :001
227: -0.000104:	*	:000000.0- :001
228: 0.000333:		:000000.0- :001
229: -0.000305:	*	:000000.0- :001
22A: 0.000125:		:000000.0- :001
22B: -0.000026:	*	:000000.0- :001
22C: 0.000065:	*	:000000.0- :001
22D: -0.000182:	*	:000000.0- :001
22E: 0.000260:		:000000.0- :001
22F: -0.000362:	*	:000000.0- :001
230: 0.000374:		:000000.0- :001
231: -0.000245:	*	:000000.0- :001
232: 0.000281:		:000000.0- :001
233: -0.000276:	*	:000000.0- :001
234: 0.000223:	*	:000000.0- :001
235: -0.000173:	*	:000000.0- :001
236: -0.000079:	*	:000000.0- :001
237: 0.000098:	*	:000000.0- :001
238: 0.000180:		:000000.0- :001
239: -0.000481:	*	:000000.0- :001
23A: 0.000359:	*	:000000.0- :001
23B: -0.000143:	*	:000000.0- :001
23C: 0.000161:		:000000.0- :001
23D: -0.000188:	*	:000000.0- :001
23E: 0.000104:	*	:000000.0- :001
23F: -0.000568:	*	:000000.0- :001
240: 0.001646:		:000000.0- :001
241: -0.000170:	*	:000000.0- :001
242: 0.000226:		:000000.0- :001
243: -0.000170:	*	:000000.0- :001
244: 0.000133:	*	:000000.0- :001
245: -0.000295:	*	:000000.0- :001
246: 0.000305:		:000000.0- :001
247: -0.000255:	*	:000000.0- :001
248: 0.000187:		:000000.0- :001
249: -0.000053:	*	:000000.0- :001
24A: 0.000090:	*	:000000.0- :001
24B: -0.000091:	*	:000000.0- :001
24C: 0.000255:		:000000.0- :001
24D: -0.000233:	*	:000000.0- :001
24E: 0.000108:		:000000.0- :001
24F: -0.000056:	*	:000000.0- :001
250: 0.000273:		:000000.0- :001

88-280013

270365-96

DNL Error, SN = 4130 (Continued)

251: -0.000353:	* :520000.0- :085
252: 0.000194:	* :100000.0- :085
253: -0.000192:	* :100000.0- :085
254: 0.000350:	* :000000.0- :085
255: -0.000258:	* :000000.0- :100
256: 0.000149:	* :000000.0- :100
257: -0.000108:	* :000000.0- :100
258: 0.000050:	* :000000.0- :100
259: -0.000120:	* :000000.0- :100
25A: -0.000085:	* :000000.0- :100
25B: 0.000087:	* :000000.0- :100
25C: 0.000130:	* :000000.0- :100
25D: -0.000192:	* :000000.0- :100
25E: 0.000002:	* :000000.0- :100
25F: -0.000069:	* :000000.0- :100
260: 0.000113:	* :000000.0- :100
261: -0.000280:	* :000000.0- :100
262: 0.000212:	* :000000.0- :100
263: -0.000129:	* :000000.0- :100
264: 0.000107:	* :000000.0- :100
265: -0.000198:	* :000000.0- :100
266: 0.000234:	* :000000.0- :100
267: -0.000203:	* :000000.0- :100
268: 0.000128:	* :000000.0- :100
269: -0.000259:	* :000000.0- :100
26A: 0.000106:	* :000000.0- :100
26B: -0.000048:	* :000000.0- :100
26C: -0.000024:	* :000000.0- :100
26D: -0.000166:	* :000000.0- :100
26E: 0.000223:	* :000000.0- :100
26F: -0.000167:	* :000000.0- :100
270: 0.000102:	* :000000.0- :100
271: -0.000196:	* :000000.0- :100
272: 0.000077:	* :000000.0- :100
273: -0.000043:	* :000000.0- :100
274: 0.000035:	* :000000.0- :100
275: -0.000272:	* :000000.0- :100
276: 0.000174:	* :000000.0- :100
277: -0.000003:	* :000000.0- :100
278: -0.000126:	* :000000.0- :100
279: -0.000103:	* :000000.0- :100
27A: 0.000109:	* :000000.0- :100
27B: -0.000063:	* :000000.0- :100
27C: 0.000216:	* :000000.0- :100
27D: -0.000465:	* :000000.0- :100
27E: 0.000379:	* :000000.0- :100
27F: -0.000262:	* :000000.0- :100
280: 0.000635:	* :000000.0- :100
281: -0.000014:	* :000000.0- :100
282: 0.000193:	* :000000.0- :100
283: -0.000476:	* :000000.0- :100
284: 0.000474:	* :000000.0- :100
285: -0.000297:	* :000000.0- :100
286: 0.000149:	* :000000.0- :100
287: -0.000166:	* :000000.0- :100
288: -0.000011:	* :000000.0- :100
289: -0.000087:	* :000000.0- :100
28A: 0.000101:	* :000000.0- :100
28B: -0.000069:	* :000000.0- :100
28C: -0.000125:	* :000000.0- :100

80-800075

270365-97

DNL Error, SN = 4130 (Continued)

28D: -0.000032:	*	162000.0	:125
28E: 0.000077:	*	161000.0	:125
28F: -0.000164:	*	1591000.0	:125
290: 0.000070:	*	1602000.0	:125
291: -0.000136:	*	1625000.0	:125
292: 0.000062:	*	1611000.0	:125
293: -0.000183:	*	1601000.0	:125
294: 0.000166:	*	160000.0	:125
295: -0.000273:	*	1611000.0	:125
296: 0.000133:	*	160000.0	:125
297: -0.000076:	*	160000.0	:125
298: 0.000045:	*	1611000.0	:125
299: -0.000234:	*	1591000.0	:125
29A: -0.000066:	*	1600000.0	:125
29B: 0.000072:	*	1600000.0	:125
29C: 0.000123:	*	1611000.0	:125
29D: -0.000304:	*	1603000.0	:125
29E: 0.000137:	*	160000.0	:125
29F: -0.000206:	*	1611000.0	:125
2A0: 0.000196:	*	1611000.0	:125
2A1: -0.000150:	*	1601000.0	:125
2A2: 0.000181:	*	160000.0	:125
2A3: -0.000160:	*	1603000.0	:125
2A4: 0.000106:	*	1611000.0	:125
2A5: -0.000262:	*	1603000.0	:125
2A6: 0.000167:	*	1611000.0	:125
2A7: -0.000123:	*	160000.0	:125
2A8: 0.000193:	*	160000.0	:125
2A9: -0.000283:	*	1601000.0	:125
2AA: -0.000117:	*	1603000.0	:125
2AB: 0.000193:	*	1611000.0	:125
2AC: 0.000140:	*	1601000.0	:125
2AD: -0.000215:	*	1601000.0	:125
2AE: -0.000044:	*	1600000.0	:125
2AF: 0.000047:	*	1600000.0	:125
2B0: 0.000028:	*	1600000.0	:125
2B1: -0.000322:	*	1603000.0	:125
2B2: 0.000207:	*	1611000.0	:125
2B3: -0.000082:	*	1600000.0	:125
2B4: 0.000125:	*	1611000.0	:125
2B5: -0.000239:	*	1601000.0	:125
2B6: 0.000078:	*	1601000.0	:125
2B7: 0.000017:	*	1600000.0	:125
2B8: 0.000128:	*	1611000.0	:125
2B9: -0.000179:	*	1600000.0	:125
2BA: 0.000101:	*	1603000.0	:125
2BB: -0.000071:	*	1603000.0	:125
2BC: 0.000359:	*	1611000.0	:125
2BD: -0.000248:	*	1611000.0	:125
2BE: 0.000054:	*	1601000.0	:125
2BF: 0.000079:	*	1603000.0	:125
2C0: 0.001402:	*	1611000.0	:125
2C1: -0.000261:	*	1603000.0	:125
2C2: 0.000127:	*	1611000.0	:125
2C3: -0.000113:	*	1601000.0	:125
2C4: 0.000268:	*	1600000.0	:125
2C5: -0.000143:	*	1600000.0	:125
2C6: 0.000007:	*	1611000.0	:125
2C7: 0.000027:	*	1600000.0	:125
2C8: 0.000243:	*	1611000.0	:125

70-288013

270365-98

DNL Error, SN = 4130 (Continued)

2C9: -0.000312:	*		1001000.0	1391
2CA: 0.000040:		*	1001000.0	1392
2CB: 0.000043:		*	1015000.0	1393
2CC: 0.000164:			1015000.0	1394
2CD: -0.000183:	*		1030000.0	1395
2CE: -0.000021:	*		1030000.0	1396
2CF: -0.000126:	*		1051000.0	1397
2D0: 0.000365:			1051000.0	1398
2D1: -0.000336:	*		1053000.0	1399
2D2: 0.000091:		*	1071000.0	1400
2D3: -0.000143:	*		1075000.0	1401
2D4: 0.000220:			1081000.0	1402
2D5: -0.000288:	*		1081000.0	1403
2D6: 0.000251:			1095000.0	1404
2D7: 0.000001:	*		1091000.0	1405
2D8: -0.000069:		*	1090000.0	1406
2D9: -0.000222:	*		1095000.0	1407
2DA: 0.000336:			111000.0	1408
2DB: -0.000071:	*		120000.0	1409
2DC: 0.000122:		*	1090000.0	1410
2DD: -0.000540:	*		1091000.0	1411
2DE: 0.000384:			1091000.0	1412
2DF: -0.000122:	*		110000.0	1413
2E0: -0.000134:	*		1021000.0	1414
2E1: -0.000114:	*		1035000.0	1415
2E2: 0.000073:		*	1090000.0	1416
2E3: -0.000040:	*		1095000.0	1417
2E4: -0.000188:	*		1071000.0	1418
2E5: 0.000033:		*	1095000.0	1419
2E6: 0.000173:		*	1071000.0	1420
2E7: -0.000049:	*		1115000.0	1421
2E8: 0.000019:	*		1000000.0	1422
2E9: -0.000234:	*		1000000.0	1423
2EA: 0.000176:		*	1001000.0	1424
2EB: -0.000281:	*		1011000.0	1425
2EC: 0.000200:		*	1000000.0	1426
2ED: -0.000174:	*		1000000.0	1427
2EE: 0.000042:		*	1001000.0	1428
2EF: -0.000086:	*		1011000.0	1429
2F0: 0.000063:		*	1001100.0	1430
2F1: -0.000160:	*		1021000.0	1431
2F2: 0.000127:		*	1000000.0	1432
2F3: -0.000134:	*		1000000.0	1433
2F4: 0.000058:		*	1020000.0	1434
2F5: -0.000211:	*		1095000.0	1435
2F6: 0.000104:		*	1031000.0	1436
2F7: -0.000031:	*		1000000.0	1437
2F8: -0.000134:	*		1000000.0	1438
2F9: -0.000083:	*		1000000.0	1439
2FA: -0.000005:	*		1011000.0	1440
2FB: -0.000026:	*		1000000.0	1441
2FC: 0.000093:		*	1050000.0	1442
2FD: -0.000015:	*		1001000.0	1443
2FE: -0.000090:	*		1000000.0	1444
2FF: 0.000570:			1040000.0	1445
300: 0.000011:	*		1051000.0	1446
301: -0.000307:	*		1052000.0	1447
302: 0.000133:		*	1055000.0	1448
303: -0.000116:	*		1055000.0	1449
304: 0.000032:	*		1057100.0	1450

QA-2880'S

270365-99

DNL Error, SN = 4130 (Continued)

```

305: -0.000166: | * | :510000.0- :035
306: 0.000148: | * | :000000.0 :035
307: -0.000214: | * | :000000.0 :035
308: 0.000217: | * | :000000.0 :035
309: -0.000323: | * | :000000.0- :035
30A: 0.000205: | * | :000000.0- :035
30B: -0.000125: | * | :000000.0- :035
30C: 0.000172: | * | :000000.0 :035
30D: -0.000224: | * | :000000.0- :035
30E: 0.000131: | * | :000000.0 :035
30F: -0.000248: | * | :000000.0- :035
310: 0.000148: | * | :000000.0 :035
311: -0.000333: | * | :000000.0- :035
312: 0.000249: | * | :000000.0 :035
313: -0.000134: | * | :000000.0 :035
314: 0.000035: | * | :000000.0- :035
315: -0.000244: | * | :000000.0- :035
316: 0.000141: | * | :000000.0 :035
317: -0.000051: | * | :000000.0- :035
318: 0.000094: | * | :000000.0 :035
319: -0.000189: | * | :000000.0- :035
31A: 0.000070: | * | :000000.0 :035
31B: -0.000012: | * | :000000.0- :035
31C: 0.000158: | * | :000000.0 :035
31D: -0.000233: | * | :000000.0- :035
31E: 0.000096: | * | :000000.0 :035
31F: -0.000286: | * | :000000.0- :035
320: 0.000173: | * | :000000.0 :035
321: -0.000229: | * | :000000.0- :035
322: 0.000137: | * | :000000.0 :035
323: -0.000112: | * | :000000.0- :035
324: 0.000086: | * | :000000.0 :035
325: -0.000205: | * | :000000.0- :035
326: 0.000183: | * | :000000.0 :035
327: -0.000116: | * | :000000.0- :035
328: -0.000054: | * | :000000.0 :035
329: -0.000096: | * | :000000.0- :035
32A: 0.000149: | * | :000000.0 :035
32B: -0.000114: | * | :000000.0- :035
32C: 0.001180: | * | :000000.0 :035
32D: -0.000156: | * | :000000.0- :035
32E: 0.000086: | * | :000000.0 :035
32F: -0.000081: | * | :000000.0- :035
330: 0.000352: | * | :000000.0 :035
331: -0.000245: | * | :000000.0- :035
332: 0.000125: | * | :000000.0 :035
333: -0.000064: | * | :000000.0- :035
334: 0.000270: | * | :000000.0 :035
335: -0.000240: | * | :000000.0- :035
336: 0.000116: | * | :000000.0 :035
337: -0.000001: | * | :000000.0- :035
338: 0.000027: | * | :000000.0 :035
339: -0.000190: | * | :000000.0- :035
33A: -0.000001: | * | :000000.0 :035
33B: 0.000048: | * | :000000.0- :035
33C: 0.000128: | * | :000000.0 :035
33D: -0.000332: | * | :000000.0- :035
33E: 0.000222: | * | :000000.0 :035
33F: -0.000265: | * | :000000.0- :035
340: 0.001727: | * | :000000.0 :035

```

270365-A0

DNL Error, SN = 4130 (Continued)

341: -0.000352:	:0*00010- :0VE
342: 0.000117:	:0S100010- :0VE
343: -0.000287:	:00*00010- :0VE
344: 0.000340:	:00100010- :00E
345: -0.000116:	:1000010- :10E
346: -0.000079:	:0000010- :00E
347: 0.000078:	:0000010- :00E
348: 0.000241:	:0100010- :00E
349: 0.000037:	:0000010- :00E
34A: -0.000181:	:111*0010- :00E
34B: -0.000080:	:0000010- :00E
34C: 0.000285:	:0100010- :00E
34D: -0.000360:	:0*00010- :00E
34E: 0.000180:	:0100010- :00E
34F: 0.000050:	:0100010- :00E
350: 0.000002:	:0000010- :00E
351: -0.000219:	:0000010- :00E
352: 0.000092:	:0000010- :00E
353: -0.000083:	:0000010- :00E
354: 0.000366:	:0000010- :00E
355: -0.000302:	:1*00010- :10E
356: 0.000188:	:0100010- :00E
357: 0.000024:	:0100010- :00E
358: -0.000125:	:0000010- :00E
359: -0.000215:	:0000010- :00E
35A: 0.000044:	:0100010- :00E
35B: 0.000026:	:0100010- :00E
35C: 0.000018:	:0100010- :00E
35D: -0.000315:	:0*00010- :00E
35E: 0.000278:	:0000010- :00E
35F: -0.000044:	:0100010- :00E
360: -0.000112:	:0000010- :00E
361: -0.000169:	:0000010- :00E
362: 0.000062:	:0100010- :00E
363: -0.000132:	:0000010- :00E
364: 0.000127:	:10100010- :10E
365: -0.000235:	:0*00010- :00E
366: 0.000109:	:0100010- :00E
367: -0.000064:	:0000010- :00E
368: 0.000157:	:0100010- :00E
369: -0.000121:	:0100010- :00E
36A: 0.000027:	:0000010- :00E
36B: -0.000109:	:0100010- :00E
36C: -0.000004:	:0100010- :00E
36D: -0.000294:	:0100010- :00E
36E: 0.000231:	:0100010- :00E
36F: -0.000030:	:0100010- :00E
370: 0.000065:	:0100010- :00E
371: -0.000268:	:0100010- :00E
372: 0.000114:	:0100010- :00E
373: -0.000050:	:0100010- :00E
374: 0.000087:	:0100010- :00E
375: -0.000210:	:0100010- :00E
376: 0.000048:	:0100010- :00E
377: 0.000111:	:0100010- :00E
378: -0.000073:	:0100010- :00E
379: -0.000186:	:0100010- :00E
37A: 0.000135:	:0100010- :00E
37B: -0.000020:	:0100010- :00E
37C: 0.000020:	:0100010- :00E

SA-280075

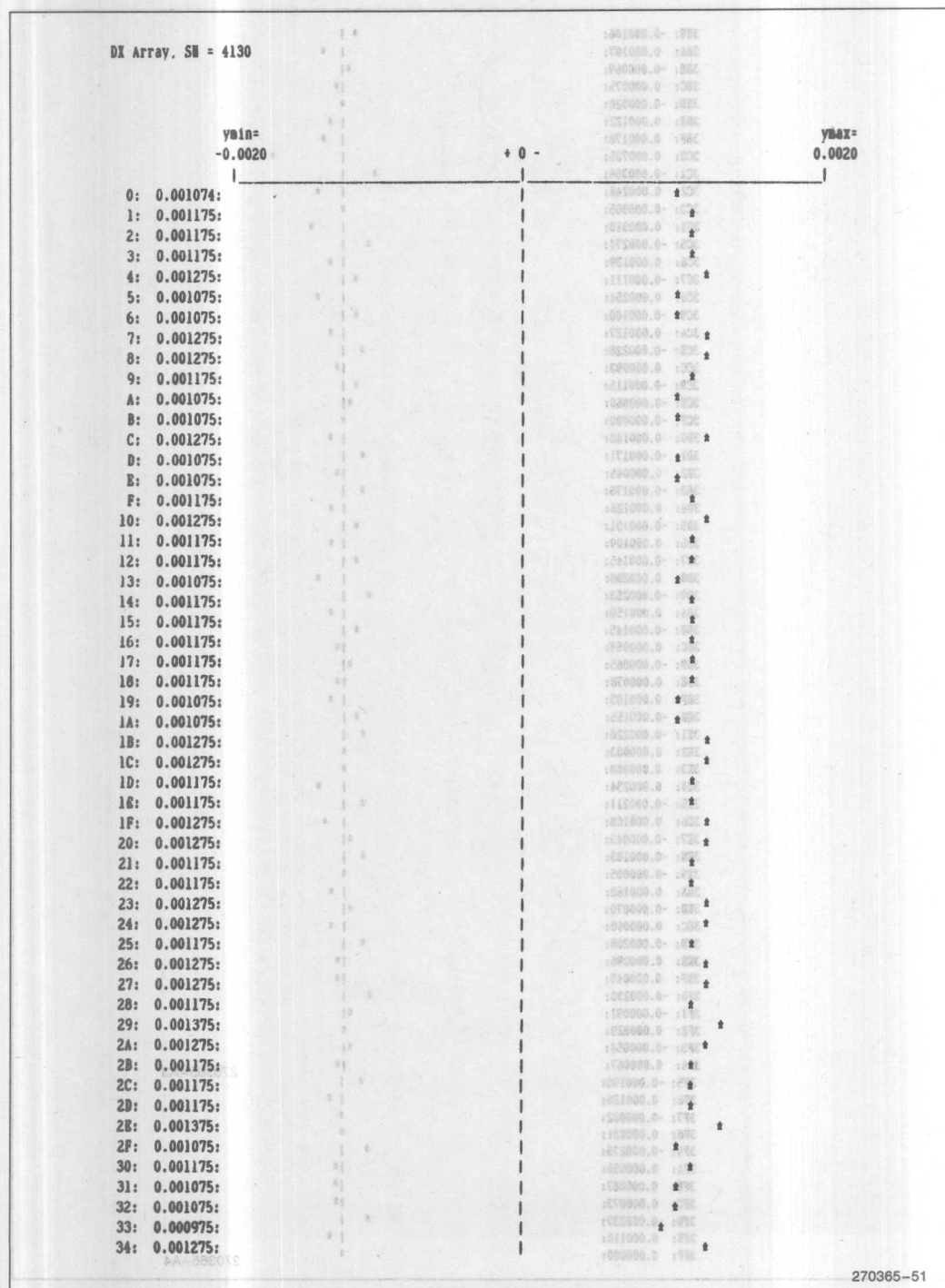
270365-A1

DNL Error, SN = 4130 (Continued)

3481 0.00000000
3482 0.00000000
3483 0.00000000
3484 0.00000000
3485 0.00000000
3486 0.00000000
3487 0.00000000
3488 0.00000000
3489 0.00000000
3490 0.00000000
3491 0.00000000
3492 0.00000000
3493 0.00000000
3494 0.00000000
3495 0.00000000
3496 0.00000000
3497 0.00000000
3498 0.00000000
3499 0.00000000
3500 0.00000000
3501 0.00000000
3502 0.00000000
3503 0.00000000
3504 0.00000000
3505 0.00000000
3506 0.00000000
3507 0.00000000
3508 0.00000000
3509 0.00000000
3510 0.00000000
3511 0.00000000
3512 0.00000000
3513 0.00000000
3514 0.00000000
3515 0.00000000
3516 0.00000000
3517 0.00000000
3518 0.00000000
3519 0.00000000
3520 0.00000000
3521 0.00000000
3522 0.00000000
3523 0.00000000
3524 0.00000000
3525 0.00000000
3526 0.00000000
3527 0.00000000
3528 0.00000000
3529 0.00000000
3530 0.00000000
3531 0.00000000
3532 0.00000000
3533 0.00000000
3534 0.00000000
3535 0.00000000
3536 0.00000000
3537 0.00000000
3538 0.00000000
3539 0.00000000
3540 0.00000000
3541 0.00000000
3542 0.00000000
3543 0.00000000
3544 0.00000000
3545 0.00000000
3546 0.00000000
3547 0.00000000
3548 0.00000000
3549 0.00000000
3550 0.00000000
3551 0.00000000
3552 0.00000000
3553 0.00000000
3554 0.00000000
3555 0.00000000
3556 0.00000000
3557 0.00000000
3558 0.00000000
3559 0.00000000
3560 0.00000000
3561 0.00000000
3562 0.00000000
3563 0.00000000
3564 0.00000000
3565 0.00000000
3566 0.00000000
3567 0.00000000
3568 0.00000000
3569 0.00000000
3570 0.00000000
3571 0.00000000
3572 0.00000000
3573 0.00000000
3574 0.00000000
3575 0.00000000
3576 0.00000000
3577 0.00000000
3578 0.00000000
3579 0.00000000
3580 0.00000000
3581 0.00000000
3582 0.00000000
3583 0.00000000
3584 0.00000000
3585 0.00000000
3586 0.00000000
3587 0.00000000
3588 0.00000000
3589 0.00000000
3590 0.00000000
3591 0.00000000
3592 0.00000000
3593 0.00000000
3594 0.00000000
3595 0.00000000
3596 0.00000000
3597 0.00000000
3598 0.00000000
3599 0.00000000
3600 0.00000000

DNL Error, SN = 4130 (Continued)

DNL Error, SN = 4130 (Continued)



Repeatability Error, SN = 4130

35:	0.001175:			:27C100.0:	11
36:	0.001075:			:27C100.0:	12
37:	0.001275:			:27C100.0:	13
38:	0.001075:			:27C100.0:	14
39:	0.001275:			:27C100.0:	15
3A:	0.001275:			:27C100.0:	16
3B:	0.001175:			:27C100.0:	17
3C:	0.001175:			:27C100.0:	18
3D:	0.001175:			:27C100.0:	19
3E:	0.001175:			:27C100.0:	1A
3F:	0.001375:			:27C100.0:	1B
40:	0.001275:			:27C100.0:	1C
41:	0.001275:			:27C100.0:	1D
42:	0.001075:			:27C100.0:	1E
43:	0.001075:			:27C100.0:	1F
44:	0.001275:			:27C100.0:	10
45:	0.001175:			:27C100.0:	11
46:	0.001175:			:27C100.0:	12
47:	0.001075:			:27C100.0:	13
48:	0.001175:			:27C100.0:	14
49:	0.001175:			:27C100.0:	15
4A:	0.001175:			:27C100.0:	16
4B:	0.001275:			:27C100.0:	17
4C:	0.001375:			:27C100.0:	18
4D:	0.001075:			:27C100.0:	19
4E:	0.001375:			:27C100.0:	1A
4F:	0.001075:			:27C100.0:	1B
50:	0.001175:			:27C100.0:	1C
51:	0.001175:			:27C100.0:	1D
52:	0.001175:			:27C100.0:	1E
53:	0.000975:			:27C100.0:	1F
54:	0.000975:			:27C100.0:	10
55:	0.001075:			:27C100.0:	11
56:	0.001175:			:27C100.0:	12
57:	0.001275:			:27C100.0:	13
58:	0.001275:			:27C100.0:	14
59:	0.001175:			:27C100.0:	15
5A:	0.001075:			:27C100.0:	16
5B:	0.001075:			:27C100.0:	17
5C:	0.001275:			:27C100.0:	18
5D:	0.001075:			:27C100.0:	19
5E:	0.001175:			:27C100.0:	1A
5F:	0.001175:			:27C100.0:	1B
60:	0.000975:			:27C100.0:	1C
61:	0.001075:			:27C100.0:	1D
62:	0.001075:			:27C100.0:	1E
63:	0.001275:			:27C100.0:	1F
64:	0.001275:			:27C100.0:	10
65:	0.001075:			:27C100.0:	11
66:	0.001175:			:27C100.0:	12
67:	0.001375:			:27C100.0:	13
68:	0.001075:			:27C100.0:	14
69:	0.001075:			:27C100.0:	15
6A:	0.001075:			:27C100.0:	16
6B:	0.001175:			:27C100.0:	17
6C:	0.001175:			:27C100.0:	18
6D:	0.001175:			:27C100.0:	19
6E:	0.001175:			:27C100.0:	1A
6F:	0.001075:			:27C100.0:	1B
70:	0.001075:			:27C100.0:	1C

88-866012

270365-52

Repeatability Error, SN = 4130 (Continued)

71:	0.001375:	1271100.0	127
72:	0.001175:	1270100.0	126
73:	0.001175:	1275100.0	127
74:	0.001175:	1270100.0	126
75:	0.001175:	1275100.0	127
76:	0.001175:	1275100.0	127
77:	0.001275:	1271100.0	127
78:	0.001175:	1271100.0	127
79:	0.001275:	1271100.0	127
7A:	0.001375:	1271100.0	127
7B:	0.001375:	1275100.0	127
7C:	0.001375:	1275100.0	127
7D:	0.001375:	1275100.0	127
7E:	0.001175:	1270100.0	126
7F:	0.001075:	1270100.0	126
80:	0.001275:	1275100.0	127
81:	0.001175:	1271100.0	127
82:	0.001275:	1271100.0	127
83:	0.001275:	1270100.0	126
84:	0.001075:	1271100.0	127
85:	0.001175:	1271100.0	127
86:	0.001175:	1271100.0	127
87:	0.001275:	1275100.0	127
88:	0.001075:	1270100.0	126
89:	0.001075:	1270100.0	126
8A:	0.001075:	1270100.0	126
8B:	0.001075:	1270100.0	126
8C:	0.001075:	1271100.0	127
8D:	0.001175:	1271100.0	127
8E:	0.001075:	1271100.0	127
8F:	0.001175:	1275100.0	127
90:	0.001175:	1275100.0	127
91:	0.001175:	1275100.0	127
92:	0.001275:	1271100.0	127
93:	0.001175:	1275100.0	127
94:	0.001075:	1275100.0	127
95:	0.001175:	1271100.0	127
96:	0.001275:	1270100.0	126
97:	0.001075:	1270100.0	126
98:	0.001175:	1275100.0	127
99:	0.001175:	1270100.0	126
9A:	0.001275:	1271100.0	127
9B:	0.001175:	1271100.0	127
9C:	0.001175:	1270000.0	126
9D:	0.001275:	1270100.0	126
9E:	0.001275:	1270100.0	126
9F:	0.001175:	1275100.0	127
A0:	0.001075:	1275100.0	127
A1:	0.001175:	1270100.0	126
A2:	0.001075:	1271100.0	127
A3:	0.001275:	1275100.0	127
A4:	0.001075:	1270100.0	126
A5:	0.001175:	1270100.0	126
A6:	0.001275:	1270100.0	126
A7:	0.001375:	1271100.0	127
A8:	0.001375:	1271100.0	127
A9:	0.001075:	1271100.0	127
AA:	0.001175:	1271100.0	127
AB:	0.001075:	1270100.0	126
AC:	0.001075:	1270100.0	126

52-366015

270365-53

Repeatability Error, SN = 4130 (Continued)

AD: 0.001075:			:27E100.0	:07
AE: 0.001175:			:27F100.0	:08
AF: 0.001075:			:27F100.0	:09
B0: 0.001075:			:27F100.0	:0A
B1: 0.001175:			:27E100.0	:0B
B2: 0.001275:			:27E100.0	:0C
B3: 0.001275:			:27E100.0	:0D
B4: 0.000975:			:27E100.0	:0E
B5: 0.001175:			:27F100.0	:0F
B6: 0.001075:			:27F100.0	:10
B7: 0.001175:			:27E100.0	:11
B8: 0.001275:			:27F100.0	:12
B9: 0.001275:			:27E100.0	:13
BA: 0.001175:			:27E100.0	:14
BB: 0.001175:			:27E100.0	:15
BC: 0.001075:			:27E100.0	:16
BD: 0.001175:			:27E100.0	:17
BE: 0.001175:			:27E100.0	:18
BF: 0.001175:			:27F100.0	:19
C0: 0.001275:			:27E100.0	:1A
C1: 0.001275:			:27E100.0	:1B
C2: 0.001075:			:27E100.0	:1C
C3: 0.000975:			:27E100.0	:1D
C4: 0.001175:			:27E100.0	:1E
C5: 0.001175:			:27F100.0	:1F
C6: 0.001175:			:27E100.0	:20
C7: 0.001275:			:27E100.0	:21
C8: 0.001175:			:27E100.0	:22
C9: 0.001175:			:27F100.0	:23
CA: 0.001075:			:27E100.0	:24
CB: 0.001375:			:27F100.0	:25
CC: 0.001275:			:27E100.0	:26
CD: 0.001275:			:27E100.0	:27
CE: 0.001175:			:27E100.0	:28
CF: 0.001275:			:27E100.0	:29
D0: 0.001175:			:27E100.0	:2A
D1: 0.001075:			:27F100.0	:2B
D2: 0.001275:			:27F100.0	:2C
D3: 0.001275:			:27E100.0	:2D
D4: 0.001175:			:27E100.0	:2E
D5: 0.001175:			:27E100.0	:2F
D6: 0.001075:			:27E100.0	:30
D7: 0.001275:			:27F100.0	:31
D8: 0.001175:			:27F100.0	:32
D9: 0.001275:			:27F100.0	:33
DA: 0.001175:			:27F100.0	:34
DB: 0.001175:			:27E100.0	:35
DC: 0.001275:			:27E100.0	:36
DD: 0.001275:			:27E100.0	:37
DE: 0.001275:			:27E100.0	:38
DF: 0.001275:			:27F100.0	:39
E0: 0.001175:			:27F100.0	:3A
E1: 0.000975:			:27F100.0	:3B
E2: 0.001275:			:27F100.0	:3C
E3: 0.001175:			:27F100.0	:3D
E4: 0.001175:			:27E100.0	:3E
E5: 0.001075:			:27F100.0	:3F
E6: 0.001175:			:27F100.0	:40
E7: 0.001175:			:27F100.0	:41
E8: 0.001175:			:27F100.0	:42

22-28803

270365-54

Repeatability Error, SN = 4130 (Continued)

B9: 0.001375:			:279100.0	:14
EA: 0.001175:			:271100.0	:15A
EB: 0.001175:			:279100.0	:15A
EC: 0.001075:			:279100.0	:16B
ED: 0.001375:			:271100.0	:16
EE: 0.001375:			:275100.0	:16
EF: 0.001275:			:275100.0	:16C
F0: 0.001175:			:279000.0	:16B
F1: 0.001175:			:271100.0	:16B
F2: 0.001175:			:279100.0	:16B
F3: 0.001275:			:271100.0	:16B
F4: 0.001175:			:275100.0	:16C
F5: 0.001275:			:275100.0	:16B
F6: 0.001075:			:271100.0	:16B
F7: 0.001375:			:271100.0	:16
F8: 0.001075:			:279100.0	:16B
F9: 0.001375:			:271100.0	:16
FA: 0.001275:			:271100.0	:16B
FB: 0.001175:			:271100.0	:16C
FC: 0.001275:			:275100.0	:16C
FD: 0.001275:			:275100.0	:16C
FE: 0.001275:			:279100.0	:16C
FF: 0.001275:			:279000.0	:16C
100: 0.001075:			:271100.0	:16C
101: 0.001175:			:271100.0	:16C
102: 0.001275:			:271100.0	:16C
103: 0.001075:			:275100.0	:16C
104: 0.001075:			:271100.0	:16C
105: 0.001175:			:271100.0	:16C
106: 0.001175:			:279100.0	:16C
107: 0.001175:			:275100.0	:16C
108: 0.001275:			:275100.0	:16C
109: 0.001375:			:275100.0	:16
10A: 0.001275:			:271100.0	:16C
10B: 0.001075:			:275100.0	:16C
10C: 0.001075:			:271100.0	:16B
10D: 0.001175:			:279100.0	:16B
10E: 0.001175:			:275100.0	:16C
10F: 0.001275:			:275100.0	:16C
110: 0.001275:			:271100.0	:16C
111: 0.001075:			:271100.0	:16B
112: 0.001075:			:279100.0	:16B
113: 0.001175:			:275100.0	:16C
114: 0.001175:			:271100.0	:16B
115: 0.001175:			:275100.0	:16C
116: 0.001175:			:271100.0	:16B
117: 0.001375:			:271100.0	:16B
118: 0.001275:			:275100.0	:16C
119: 0.001275:			:275100.0	:16C
11A: 0.001175:			:275100.0	:16B
11B: 0.001175:			:275100.0	:16C
11C: 0.001175:			:271100.0	:16B
11D: 0.001175:			:279000.0	:16B
11E: 0.001175:			:275100.0	:16B
11F: 0.001175:			:271100.0	:16B
120: 0.001275:			:271100.0	:16B
121: 0.001175:			:279100.0	:16B
122: 0.001175:			:271100.0	:16C
123: 0.001175:			:271100.0	:16B
124: 0.001175:			:271100.0	:16C

43-395073

270365-55

Repeatability Error, SN = 4130 (Continued)

125:	0.001175:	125:	0.001175:
126:	0.001175:	126:	0.001175:
127:	0.001275:	127:	0.001275:
128:	0.001075:	128:	0.001075:
129:	0.001175:	129:	0.001175:
12A:	0.001275:	12A:	0.001275:
12B:	0.001175:	12B:	0.001175:
12C:	0.001175:	12C:	0.001175:
12D:	0.001275:	12D:	0.001275:
12E:	0.001075:	12E:	0.001075:
12F:	0.001275:	12F:	0.001275:
130:	0.001175:	130:	0.001175:
131:	0.001175:	131:	0.001175:
132:	0.001075:	132:	0.001075:
133:	0.001075:	133:	0.001075:
134:	0.001275:	134:	0.001275:
135:	0.001275:	135:	0.001275:
136:	0.001275:	136:	0.001275:
137:	0.001075:	137:	0.001075:
138:	0.001175:	138:	0.001175:
139:	0.001275:	139:	0.001275:
13A:	0.001175:	13A:	0.001175:
13B:	0.001275:	13B:	0.001275:
13C:	0.001175:	13C:	0.001175:
13D:	0.001175:	13D:	0.001175:
13E:	0.001275:	13E:	0.001275:
13F:	0.001075:	13F:	0.001075:
140:	0.001075:	140:	0.001075:
141:	0.001075:	141:	0.001075:
142:	0.001075:	142:	0.001075:
143:	0.001075:	143:	0.001075:
144:	0.001175:	144:	0.001175:
145:	0.001275:	145:	0.001275:
146:	0.001375:	146:	0.001375:
147:	0.001375:	147:	0.001375:
148:	0.001275:	148:	0.001275:
149:	0.001275:	149:	0.001275:
14A:	0.001275:	14A:	0.001275:
14B:	0.001275:	14B:	0.001275:
14C:	0.001375:	14C:	0.001375:
14D:	0.001375:	14D:	0.001375:
14E:	0.001175:	14E:	0.001175:
14F:	0.001175:	14F:	0.001175:
150:	0.001175:	150:	0.001175:
151:	0.001275:	151:	0.001275:
152:	0.001375:	152:	0.001375:
153:	0.001275:	153:	0.001275:
154:	0.001275:	154:	0.001275:
155:	0.001175:	155:	0.001175:
156:	0.001175:	156:	0.001175:
157:	0.001275:	157:	0.001275:
158:	0.001175:	158:	0.001175:
159:	0.001175:	159:	0.001175:
15A:	0.001175:	15A:	0.001175:
15B:	0.001275:	15B:	0.001275:
15C:	0.001175:	15C:	0.001175:
15D:	0.001375:	15D:	0.001375:
15E:	0.001275:	15E:	0.001275:
15F:	0.001375:	15F:	0.001375:
160:	0.001375:	160:	0.001375:

12-286073

270365-56

Repeatability Error, SN = 4130 (Continued)

Repeatability Error, SN = 4130 (Continued)

19D: 0.001275:			1275100.0 *	1901
19E: 0.001275:			1275100.0 *	1901
19F: 0.001275:			1275100.0 *	1901
1A0: 0.001275:			1275100.0 *	1901
1A1: 0.001275:			1275100.0 *	1901
1A2: 0.001375:			1275100.0 *	1901
1A3: 0.001475:			1275100.0 *	1901
1A4: 0.001275:			1275100.0 *	1901
1A5: 0.001075:			1275100.0 *	1901
1A6: 0.001175:			1275100.0 *	1901
1A7: 0.001275:			1275100.0 *	1901
1A8: 0.001175:			1275100.0 *	1901
1A9: 0.001375:			1275100.0 *	1901
1AA: 0.001275:			1275100.0 *	1901
1AB: 0.001275:			1275100.0 *	1901
1AC: 0.001175:			1275100.0 *	1901
1AD: 0.001175:			1275100.0 *	1901
1AE: 0.001175:			1275100.0 *	1901
1AF: 0.001175:			1275100.0 *	1901
1B0: 0.001075:			1275100.0 *	1901
1B1: 0.001375:			1275100.0 *	1901
1B2: 0.000975:			1275100.0 *	1901
1B3: 0.001175:			1275100.0 *	1901
1B4: 0.001075:			1275100.0 *	1901
1B5: 0.001375:			1275100.0 *	1901
1B6: 0.001375:			1275100.0 *	1901
1B7: 0.001175:			1275100.0 *	1901
1B8: 0.001175:			1275100.0 *	1901
1B9: 0.001375:			1275100.0 *	1901
1BA: 0.001175:			1275100.0 *	1901
1BB: 0.001475:			1275100.0 *	1901
1BC: 0.001175:			1275100.0 *	1901
1BD: 0.001275:			1275100.0 *	1901
1BE: 0.001175:			1275100.0 *	1901
1BF: 0.001175:			1275100.0 *	1901
1C0: 0.001175:			1275100.0 *	1901
1C1: 0.001175:			1275100.0 *	1901
1C2: 0.001175:			1275100.0 *	1901
1C3: 0.001275:			1275100.0 *	1901
1C4: 0.001375:			1275100.0 *	1901
1C5: 0.001275:			1275100.0 *	1901
1C6: 0.001175:			1275100.0 *	1901
1C7: 0.001175:			1275100.0 *	1901
1C8: 0.001375:			1275100.0 *	1901
1C9: 0.001175:			1275100.0 *	1901
1CA: 0.001075:			1275100.0 *	1901
1CB: 0.001075:			1275100.0 *	1901
1CC: 0.001275:			1275100.0 *	1901
1CD: 0.001375:			1275100.0 *	1901
1CE: 0.001275:			1275100.0 *	1901
1CF: 0.001075:			1275100.0 *	1901
1D0: 0.001175:			1275100.0 *	1901
1D1: 0.001275:			1275100.0 *	1901
1D2: 0.001275:			1275100.0 *	1901
1D3: 0.001275:			1275100.0 *	1901
1D4: 0.001275:			1275100.0 *	1901
1D5: 0.001275:			1275100.0 *	1901
1D6: 0.001275:			1275100.0 *	1901
1D7: 0.001175:			1275100.0 *	1901
1D8: 0.001275:			1275100.0 *	1901

23-382073

270365-58

Repeatability Error, SN = 4130 (Continued)

1D9:	0.001275:			12VS100.0	1291
1DA:	0.001275:			12VS100.0	1291
1DB:	0.001175:			12VS100.0	1291
1DC:	0.001175:			12VS100.0	1291
1DD:	0.001275:			12VS100.0	1291
1DE:	0.001175:			12VS100.0	1291
1DF:	0.001375:			12VS100.0	1291
1E0:	0.001375:			12VS100.0	1291
1E1:	0.001075:			12VS100.0	1291
1E2:	0.001275:			12VS100.0	1291
1E3:	0.001175:			12VS100.0	1291
1E4:	0.001175:			12VS100.0	1291
1E5:	0.001275:			12VS100.0	1291
1E6:	0.001175:			12VS100.0	1291
1E7:	0.001275:			12VS100.0	1291
1E8:	0.001175:			12VS100.0	1291
1E9:	0.001175:			12VS100.0	1291
1EA:	0.001175:			12VS100.0	1291
1EB:	0.001075:			12VS100.0	1291
1EC:	0.001275:			12VS100.0	1291
1ED:	0.001275:			12VS100.0	1291
1EE:	0.001175:			12VS100.0	1291
1EF:	0.001275:			12VS100.0	1291
1F0:	0.001375:			12VS100.0	1291
1F1:	0.001075:			12VS100.0	1291
1F2:	0.001175:			12VS100.0	1291
1F3:	0.001574:			12VS100.0	1291
1F4:	0.001275:			12VS100.0	1291
1F5:	0.001175:			12VS100.0	1291
1F6:	0.001175:			12VS100.0	1291
1F7:	0.001275:			12VS100.0	1291
1F8:	0.001175:			12VS100.0	1291
1F9:	0.001175:			12VS100.0	1291
1FA:	0.001175:			12VS100.0	1291
1FB:	0.001275:			12VS100.0	1291
1FC:	0.001175:			12VS100.0	1291
1FD:	0.001175:			12VS100.0	1291
1FE:	0.001275:			12VS100.0	1291
1FF:	0.001275:			12VS100.0	1291
200:	0.001275:			12VS100.0	1291
201:	0.001375:			12VS100.0	1291
202:	0.001375:			12VS100.0	1291
203:	0.001175:			12VS100.0	1291
204:	0.001175:			12VS100.0	1291
205:	0.001275:			12VS100.0	1291
206:	0.001175:			12VS100.0	1291
207:	0.001375:			12VS100.0	1291
208:	0.001275:			12VS100.0	1291
209:	0.001175:			12VS100.0	1291
20A:	0.001175:			12VS100.0	1291
20B:	0.001275:			12VS100.0	1291
20C:	0.001275:			12VS100.0	1291
20D:	0.001275:			12VS100.0	1291
20E:	0.001175:			12VS100.0	1291
20F:	0.001175:			12VS100.0	1291
210:	0.001175:			12VS100.0	1291
211:	0.001175:			12VS100.0	1291
212:	0.001075:			12VS100.0	1291
213:	0.001275:			12VS100.0	1291
214:	0.001175:			12VS100.0	1291

88-3050YS

270365-59

Repeatability Error, SN = 4130 (Continued)

251: 0.001175:			127S100.0 * 115
252: 0.001275:			127S100.0 * 115
253: 0.001275:			127S100.0 * 115
254: 0.001175:			127S100.0 * 115
255: 0.001375:			127S100.0 * 115
256: 0.001275:			127S100.0 * 115
257: 0.001275:			127S100.0 * 115
258: 0.001275:			127S100.0 * 115
259: 0.001175:			127S100.0 * 115
25A: 0.001075:			127S100.0 * 115
25B: 0.001175:			127S100.0 * 115
25C: 0.001475:			127S100.0 * 115
25D: 0.001275:			127S100.0 * 115
25E: 0.001275:			127S100.0 * 115
25F: 0.001175:			127S100.0 * 115
260: 0.001275:			127S100.0 * 115
261: 0.001175:			127S100.0 * 115
262: 0.001175:			127S100.0 * 115
263: 0.001275:			127S100.0 * 115
264: 0.001275:			127S100.0 * 115
265: 0.001275:			127S100.0 * 115
266: 0.001175:			127S100.0 * 115
267: 0.001375:			127S100.0 * 115
268: 0.001275:			127S100.0 * 115
269: 0.001275:			127S100.0 * 115
26A: 0.001175:			127S100.0 * 115
26B: 0.001175:			127S100.0 * 115
26C: 0.001375:			127S100.0 * 115
26D: 0.001375:			127S100.0 * 115
26E: 0.001375:			127S100.0 * 115
26F: 0.001475:			127S100.0 * 115
270: 0.001175:			127S100.0 * 115
271: 0.001375:			127S100.0 * 115
272: 0.001175:			127S100.0 * 115
273: 0.001075:			127S100.0 * 115
274: 0.001275:			127S100.0 * 115
275: 0.001175:			127S100.0 * 115
276: 0.001275:			127S100.0 * 115
277: 0.001375:			127S100.0 * 115
278: 0.001375:			127S100.0 * 115
279: 0.001375:			127S100.0 * 115
27A: 0.001275:			127S100.0 * 115
27B: 0.001275:			127S100.0 * 115
27C: 0.001275:			127S100.0 * 115
27D: 0.001175:			127S100.0 * 115
27E: 0.001075:			127S100.0 * 115
27F: 0.001275:			127S100.0 * 115
280: 0.001275:			127S100.0 * 115
281: 0.001375:			127S100.0 * 115
282: 0.001275:			127S100.0 * 115
283: 0.001275:			127S100.0 * 115
284: 0.001275:			127S100.0 * 115
285: 0.001375:			127S100.0 * 115
286: 0.001275:			127S100.0 * 115
287: 0.001375:			127S100.0 * 115
288: 0.001175:			127S100.0 * 115
289: 0.001275:			127S100.0 * 115
28A: 0.001175:			127S100.0 * 115
28B: 0.001375:			127S100.0 * 115
28C: 0.001175:			127S100.0 * 115

00-200075

270365-61

Repeatability Error, SN = 4130 (Continued)

28D: 0.001375:			127E100.0 1085
28E: 0.001175:			127E100.0 1085
28F: 0.001375:			127E100.0 1085
290: 0.001275:			127E100.0 1085
291: 0.001275:			127E100.0 1085
292: 0.001175:			127E100.0 1085
293: 0.001175:			127E100.0 1085
294: 0.001175:			127E100.0 1085
295: 0.001175:			127E100.0 1085
296: 0.001275:			127E100.0 1085
297: 0.001375:			127E100.0 1085
298: 0.001275:			127E100.0 1085
299: 0.001375:			127E100.0 1085
29A: 0.001375:			127E100.0 1085
29B: 0.001375:			127E100.0 1085
29C: 0.001275:			127E100.0 1085
29D: 0.001175:			127E100.0 1085
29E: 0.001375:			127E100.0 1085
29F: 0.001175:			127E100.0 1085
2A0: 0.001175:			127E100.0 1085
2A1: 0.001175:			127E100.0 1085
2A2: 0.001375:			127E100.0 1085
2A3: 0.001275:			127E100.0 1085
2A4: 0.001175:			127E100.0 1085
2A5: 0.001175:			127E100.0 1085
2A6: 0.001175:			127E100.0 1085
2A7: 0.001275:			127E100.0 1085
2A8: 0.001475:			127E100.0 1085
2A9: 0.001275:			127E100.0 1085
2AA: 0.001175:			127E100.0 1085
2AB: 0.001275:			127E100.0 1085
2AC: 0.001375:			127E100.0 1085
2AD: 0.001275:			127E100.0 1085
2AE: 0.001275:			127E100.0 1085
2AF: 0.001175:			127E100.0 1085
2B0: 0.001175:			127E100.0 1085
2B1: 0.001175:			127E100.0 1085
2B2: 0.001175:			127E100.0 1085
2B3: 0.001275:			127E100.0 1085
2B4: 0.001175:			127E100.0 1085
2B5: 0.001275:			127E100.0 1085
2B6: 0.001175:			127E100.0 1085
2B7: 0.001275:			127E100.0 1085
2B8: 0.001275:			127E100.0 1085
2B9: 0.001275:			127E100.0 1085
2BA: 0.001275:			127E100.0 1085
2BB: 0.001375:			127E100.0 1085
2BC: 0.001275:			127E100.0 1085
2BD: 0.001375:			127E100.0 1085
2BE: 0.001375:			127E100.0 1085
2BF: 0.001375:			127E100.0 1085
2C0: 0.001375:			127E100.0 1085
2C1: 0.001375:			127E100.0 1085
2C2: 0.001375:			127E100.0 1085
2C3: 0.001375:			127E100.0 1085
2C4: 0.001375:			127E100.0 1085
2C5: 0.001475:			127E100.0 1085
2C6: 0.001275:			127E100.0 1085
2C7: 0.001375:			127E100.0 1085
2C8: 0.001275:			127E100.0 1085

03-2800VS

270365-62

Repeatability Error, SN = 4130 (Continued)

2C9:	0.001375:			:270100.0	*:8
2CA:	0.001175:			:270100.0	*:805
2CB:	0.001275:			:270100.0	*:805
2CC:	0.001275:			:270100.0	*:805
2CD:	0.001475:			:270100.0	*:805
2CE:	0.001275:			:270100.0	*:805
2CF:	0.001175:			:270100.0	*:805
2D0:	0.001175:			:270100.0	*:805
2D1:	0.001175:			:270100.0	*:805
2D2:	0.001275:			:270100.0	*:805
2D3:	0.001375:			:270100.0	*:805
2D4:	0.001175:			:270100.0	*:805
2D5:	0.001175:			:270100.0	*:805
2D6:	0.001275:			:270100.0	*:805
2D7:	0.001375:			:270100.0	*:805
2D8:	0.001275:			:270100.0	*:805
2D9:	0.001275:			:270100.0	*:805
2DA:	0.001475:			:270100.0	*:805
2DB:	0.001475:			:270100.0	*:805
2DC:	0.001375:			:270100.0	*:805
2DD:	0.001375:			:270100.0	*:805
2DE:	0.001375:			:270100.0	*:805
2DF:	0.001375:			:270100.0	*:805
2E0:	0.001175:			:270100.0	*:805
2E1:	0.001375:			:270100.0	*:805
2E2:	0.001275:			:270100.0	*:805
2E3:	0.001175:			:270100.0	*:805
2E4:	0.001175:			:270100.0	*:805
2E5:	0.001275:			:270100.0	*:805
2E6:	0.001275:			:270100.0	*:805
2E7:	0.001375:			:270100.0	*:805
2E8:	0.001175:			:270100.0	*:805
2E9:	0.001275:			:270100.0	*:805
2EA:	0.001275:			:270100.0	*:805
2EB:	0.001175:			:270100.0	*:805
2EC:	0.001375:			:270100.0	*:805
2ED:	0.001275:			:270100.0	*:805
2EE:	0.001275:			:270100.0	*:805
2EF:	0.001175:			:270100.0	*:805
2F0:	0.001275:			:270100.0	*:805
2F1:	0.001375:			:270100.0	*:805
2F2:	0.001375:			:270100.0	*:805
2F3:	0.001275:			:270100.0	*:805
2F4:	0.001275:			:270100.0	*:805
2F5:	0.001375:			:270100.0	*:805
2F6:	0.001475:			:270100.0	*:805
2F7:	0.001375:			:270100.0	*:805
2F8:	0.001375:			:270100.0	*:805
2F9:	0.001475:			:270100.0	*:805
2FA:	0.001275:			:270100.0	*:805
2FB:	0.001175:			:270100.0	*:805
2FC:	0.001275:			:270100.0	*:805
2FD:	0.001275:			:270100.0	*:805
2FE:	0.001275:			:270100.0	*:805
2FF:	0.001275:			:270100.0	*:805
300:	0.001075:			:270100.0	*:805
301:	0.001275:			:270100.0	*:805
302:	0.001375:			:270100.0	*:805
303:	0.001275:			:270100.0	*:805
304:	0.001175:			:270100.0	*:805

SD-550075

270365-63

Repeatability Error, SN = 4130 (Continued)

305: 0.001475:			127C100.0 11A
306: 0.001275:			127C100.0 11B
307: 0.001375:			127C100.0 11C
308: 0.001375:			127C100.0 11D
309: 0.001275:			127C100.0 11E
30A: 0.001275:			127C100.0 11F
30B: 0.001375:			127C100.0 11G
30C: 0.001275:			127C100.0 11H
30D: 0.001475:			127C100.0 11I
30E: 0.001275:			127C100.0 11J
30F: 0.001375:			127C100.0 11K
310: 0.001175:			127C100.0 11L
311: 0.001175:			127C100.0 11M
312: 0.001275:			127C100.0 11N
313: 0.001275:			127C100.0 11O
314: 0.001375:			127C100.0 11P
315: 0.001275:			127C100.0 11Q
316: 0.001275:			127C100.0 11R
317: 0.001275:			127C100.0 11S
318: 0.001175:			127C100.0 11T
319: 0.001375:			127C100.0 11U
31A: 0.001275:			127C100.0 11V
31B: 0.001075:			127C100.0 11W
31C: 0.001175:			127C100.0 11X
31D: 0.001375:			127C100.0 11Y
31E: 0.001375:			127C100.0 11Z
31F: 0.001375:			127C100.0 11A
320: 0.001375:			127C100.0 11B
321: 0.001375:			127C100.0 11C
322: 0.001375:			127C100.0 11D
323: 0.001275:			127C100.0 11E
324: 0.001174:			127C100.0 11F
325: 0.001575:			127C100.0 11G
326: 0.001275:			127C100.0 11H
327: 0.001475:			127C100.0 11I
328: 0.001174:			127C100.0 11J
329: 0.001375:			127C100.0 11K
32A: 0.001275:			127C100.0 11L
32B: 0.001275:			127C100.0 11M
32C: 0.001375:			127C100.0 11N
32D: 0.001375:			127C100.0 11O
32E: 0.001375:			127C100.0 11P
32F: 0.001375:			127C100.0 11Q
330: 0.001174:			127C100.0 11R
331: 0.001174:			127C100.0 11S
332: 0.001475:			127C100.0 11T
333: 0.001275:			127C100.0 11U
334: 0.001275:			127C100.0 11V
335: 0.001575:			127C100.0 11W
336: 0.001475:			127C100.0 11X
337: 0.001174:			127C100.0 11Y
338: 0.001275:			127C100.0 11Z
339: 0.001275:			127C100.0 11A
33A: 0.001275:			127C100.0 11B
33B: 0.001275:			127C100.0 11C
33C: 0.001375:			127C100.0 11D
33D: 0.001375:			127C100.0 11E
33E: 0.001275:			127C100.0 11F
33F: 0.001275:			127C100.0 11G
340: 0.001174:			127C100.0 11H

25-2800TS

270365-64

Repeatability Error, SN = 4130 (Continued)

341: 0.001375:			:273100.0	* 100
342: 0.001375:			:273100.0	* 100
343: 0.001375:			:273100.0	* 100
344: 0.001174:			:273100.0	* 100
345: 0.001275:			:273100.0	* 100
346: 0.001174:			:273100.0	* 100
347: 0.001575:			:273100.0	* 100
348: 0.001375:			:273100.0	* 100
349: 0.001275:			:273100.0	* 100
34A: 0.001174:			:273100.0	* 100
34B: 0.001275:			:273100.0	* 100
34C: 0.001275:			:273100.0	* 100
34D: 0.001275:			:273100.0	* 100
34E: 0.001275:			:273100.0	* 100
34F: 0.001275:			:273100.0	* 100
350: 0.001375:			:273100.0	* 100
351: 0.001375:			:273100.0	* 100
352: 0.001375:			:273100.0	* 100
353: 0.001375:			:273100.0	* 100
354: 0.001275:			:273100.0	* 100
355: 0.001375:			:273100.0	* 100
356: 0.001375:			:273100.0	* 100
357: 0.001375:			:273100.0	* 100
358: 0.001074:			:273100.0	* 100
359: 0.001275:			:273100.0	* 100
35A: 0.001074:			:273100.0	* 100
35B: 0.001375:			:273100.0	* 100
35C: 0.001375:			:273100.0	* 100
35D: 0.001375:			:273100.0	* 100
35E: 0.001275:			:273100.0	* 100
35F: 0.001375:			:273100.0	* 100
360: 0.001174:			:273100.0	* 100
361: 0.001375:			:273100.0	* 100
362: 0.001275:			:273100.0	* 100
363: 0.001275:			:273100.0	* 100
364: 0.001275:			:273100.0	* 100
365: 0.001375:			:273100.0	* 100
366: 0.001375:			:273100.0	* 100
367: 0.001375:			:273100.0	* 100
368: 0.001275:			:273100.0	* 100
369: 0.001174:			:273100.0	* 100
36A: 0.001275:			:273100.0	* 100
36B: 0.001375:			:273100.0	* 100
36C: 0.001375:			:273100.0	* 100
36D: 0.001275:			:273100.0	* 100
36E: 0.001275:			:273100.0	* 100
36F: 0.001475:			:273100.0	* 100
370: 0.001375:			:273100.0	* 100
371: 0.001275:			:273100.0	* 100
372: 0.001375:			:273100.0	* 100
373: 0.001375:			:273100.0	* 100
374: 0.001275:			:273100.0	* 100
375: 0.001275:			:273100.0	* 100
376: 0.001275:			:273100.0	* 100
377: 0.001275:			:273100.0	* 100
378: 0.001275:			:273100.0	* 100
379: 0.001575:			:273100.0	* 100
37A: 0.001475:			:273100.0	* 100
37B: 0.001375:			:273100.0	* 100
37C: 0.001275:			:273100.0	* 100

18-2800TS

270365-65

Repeatability Error, SN = 4130 (Continued)

37D: 0.001375:	1275100.0	1275100.0
37E: 0.001375:	1275100.0	1275100.0
37F: 0.001375:	1275100.0	1275100.0
380: 0.001475:	1275100.0	1275100.0
381: 0.001475:	1275100.0	1275100.0
382: 0.001275:	1275100.0	1275100.0
383: 0.001475:	1275100.0	1275100.0
384: 0.001375:	1275100.0	1275100.0
385: 0.001475:	1275100.0	1275100.0
386: 0.001275:	1275100.0	1275100.0
387: 0.001275:	1275100.0	1275100.0
388: 0.001375:	1275100.0	1275100.0
389: 0.001174:	1275100.0	1275100.0
38A: 0.001275:	1275100.0	1275100.0
38B: 0.001275:	1275100.0	1275100.0
38C: 0.001275:	1275100.0	1275100.0
38D: 0.001275:	1275100.0	1275100.0
38E: 0.001275:	1275100.0	1275100.0
38F: 0.001275:	1275100.0	1275100.0
390: 0.001275:	1275100.0	1275100.0
391: 0.001174:	1275100.0	1275100.0
392: 0.001375:	1275100.0	1275100.0
393: 0.001375:	1275100.0	1275100.0
394: 0.001375:	1275100.0	1275100.0
395: 0.001375:	1275100.0	1275100.0
396: 0.001074:	1275100.0	1275100.0
397: 0.001275:	1275100.0	1275100.0
398: 0.001375:	1275100.0	1275100.0
399: 0.001375:	1275100.0	1275100.0
39A: 0.001275:	1275100.0	1275100.0
39B: 0.001375:	1275100.0	1275100.0
39C: 0.001275:	1275100.0	1275100.0
39D: 0.001275:	1275100.0	1275100.0
39E: 0.001475:	1275100.0	1275100.0
39F: 0.001375:	1275100.0	1275100.0
3A0: 0.001275:	1275100.0	1275100.0
3A1: 0.001275:	1275100.0	1275100.0
3A2: 0.001275:	1275100.0	1275100.0
3A3: 0.001275:	1275100.0	1275100.0
3A4: 0.001375:	1275100.0	1275100.0
3A5: 0.001275:	1275100.0	1275100.0
3A6: 0.001174:	1275100.0	1275100.0
3A7: 0.000974:	1275100.0	1275100.0
3A8: 0.001475:	1275100.0	1275100.0
3A9: 0.001375:	1275100.0	1275100.0
3AA: 0.001275:	1275100.0	1275100.0
3AB: 0.001475:	1275100.0	1275100.0
3AC: 0.001275:	1275100.0	1275100.0
3AD: 0.001375:	1275100.0	1275100.0
3AE: 0.001375:	1275100.0	1275100.0
3AF: 0.001375:	1275100.0	1275100.0
3B0: 0.001475:	1275100.0	1275100.0
3B1: 0.001174:	1275100.0	1275100.0
3B2: 0.001174:	1275100.0	1275100.0
3B3: 0.001275:	1275100.0	1275100.0
3B4: 0.001275:	1275100.0	1275100.0
3B5: 0.001275:	1275100.0	1275100.0
3B6: 0.001275:	1275100.0	1275100.0
3B7: 0.001375:	1275100.0	1275100.0
3B8: 0.001375:	1275100.0	1275100.0

Repeatability Error, SN = 4130 (Continued)

270365-66

Repeatability Error, SN = 4130 (Continued)

APPENDIX E BIBLIOGRAPHY

- A/D Processing with Microcontrollers, Katausky, Horden, Smith
- IEEE STD. 746-1984
- Intel Application Note AP-124 - High-Speed Digital Servos for Motor Control Using the 2920/21 Signal Processor
- Intel Application Note AP-125 - Designing Microcontroller Systems for Electrically Noisy Environments
- Apfel, R., et. al., "Signal-Processing Chips enrich telephone line- card Architecture". Electronics, May 5, 1982.
- Analog Devices - Data-Acquisition Databook 1984, Volume 1
- Blahut, Richard E., "Fast algorithms for digital signal processing", Addison Wesley Publishing Company, Inc., 1985.
- Boyes, ed. - Syncro and Resolver Conversion, 1980
- Brown, Robert Grover, "Introduction to random signal analysis and Kalman filtering". John Wiley & Sons, Inc., 1983.
- Burr-Brown Application Note, Testing of Analog-to-Digital Converters
- Burton and Dexter - Microprocessor Systems Handbook, 1977
- Candy, J., et. al., "The Structure of Quantization Noise from Sigma-Delta Modulation", IEEE Transaction on Comm. Vol. Com. 29, No. 9, Sept. 1981.
- Candy, J., et. al., "Using Triangularly Weighted Interpolation to Get 13-Bit PCM from a Sigma-Delta Modulator", IEEE Transaction on Comm., Nov. 1976.
- Electronic Analog-to-Digital Converters, Seitzer, Pretzl, Handy
- Handbook of Electronic Calculations, Chapter 15, Analog-Digital Conversion
- Harris Analog and Telecom Data Book
- IEEE 162
- Irwensen, J., "Calculated Quantization Noise of Single - Integration Delta Modulation Coders" BSTJ Sept. 1969.
- ITT Digital 2000 VLSI Digital TV System, MAA 2300 Audio A/D Converter, Edition 1983/9.
- MIL-M-38510/135 June 4, 1984
- MIL-M-38510/135 May 6, 1985
- Modern Electronic Circuits Reference Manual
- NBS Staff Reports, May/June 1981 P.22/23
- Sheingold, ed. - Analog-Digital Conversion Handbook, 1972
- Sheingold, ed. - Analog-Digital Conversion Notes, 1977
- Sheingold, ed. - Non-Linear Circuits Handbook, 1974
- Sheingold, ed. - Transducer Interfacing Handbook, 1980
- Steele, R., "Delta Modulation Systems", Pentech Press Limited, 1975.
- Taylor, Fred U., "Digital filter design handbook", Marcel Dekker, Inc., 1983.
- Terminology Related to the Perf of S/H, A/D, D/A Circuits, IEEE Transactions

Many applications have throughput time requirements on the order of a few hundred milliseconds, and don't require real-time image analysis.

A Single-Chip Image Processor

A.L. Pai and S.H. Lin, Arizona State University, and David P. Ryan, Intel Corporation

Most of the research efforts on image processing focus on expanding the complexity and dimension of image analysis. Unfortunately, this emphasis results in algorithms that are so computationally intensive that expensive special-purpose vector and pipeline processors are required to evaluate an image fast enough to be considered "real-time." Not all applications, however, have the burdensome requirements of true real-life image analysis. Specifically, applications that have image throughput time requirements of greater than a few hundred milliseconds can use a lower cost, general-purpose microprocessor-based system. Applications that have even slower frame rates are candidates for not only the use of lower cost CPUs, but also allow for replacement of video-rate flash A/D converters with slower, less expensive converters.

Addressing the most cost-sensitive applications, the design described herein uses Intel's 16-bit microcontroller to implement a single-chip image processor. The on-chip 10-bit A/D converter of the controller digitizes the image of a charge injection device (CID) camera, while the chip's 16-bit CPU executes a library of standard vision algorithms and reports the results by passing a few tokens over an on-chip universal asynchronous receiver-transmitter (UART).

SYSTEM OVERVIEW

A block diagram of the single-chip im-

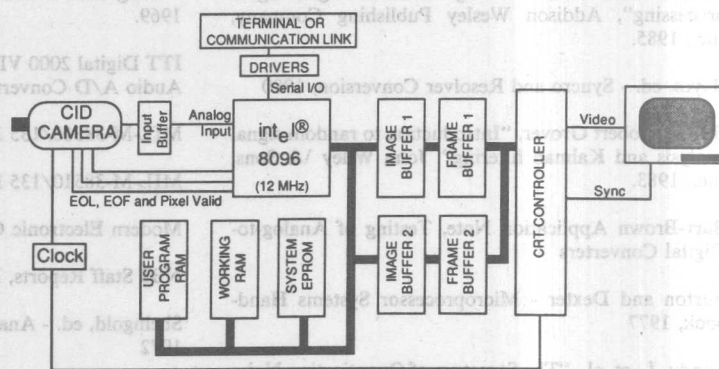


Figure 1. System block diagram.

age processor is shown in Figure 1. The image is acquired by the CID camera and input as an analog voltage to the 8096 where it is digitized and stored in one of two image buffers. The digital image is stored as an $N \times N$ matrix of 8-bit values corresponding to the gray level intensity at each picture element (pixel) as shown in Figure 2.

After an image resides in an image buffer, the 8096 can execute a number of standard image processing algorithms available as system monitor commands. These programs perform thresholding and filtering functions on the digital image, and can analyze objects found within the image. If the 8096 were attached to a host system instead of a terminal, custom pro-

grams could be downloaded to the user program RAM and executed.

To view the raw and processed images, a CRT controller is used to keep a video monitor updated with the images stored in the two frame buffer memories of Figure 1. The 8096 updates the frame buffers with the data in its image buffers depending upon commands given to the system.

► **Hardware.** The system is composed of a 128 x 128 CID camera and Intel's 8096 (with on-chip A/D) for image acquisition and analysis. A standard CRT controller was added for displaying raw and processed images as directed by the 8096. Driving the decision to use a 128 by 128 digital

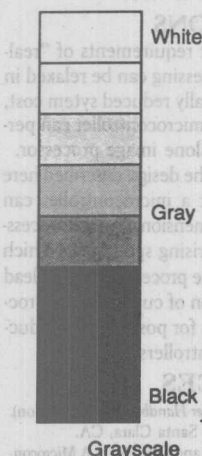
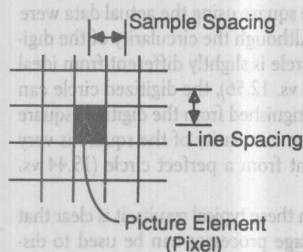
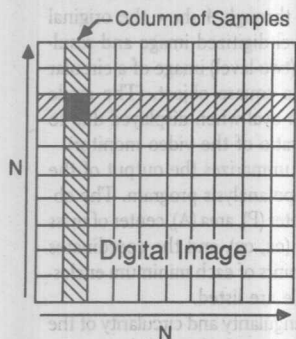


Figure 2. Representation of an $N \times N$ digital image.

image was the desire to store and operate upon two images simultaneously while minimizing memory requirements.

The image processing and communications software takes 5K of the 8K bytes allocated to the system monitor space, and would fit in the on-chip ROM space of an

8397 with 3K left. Two 32K x 8 SRAMs are used to provide space for two 16K byte image buffers, a 16K section of working RAM, and space for user-downloadable programs that are invoked by the monitor.

Two 16K byte frame-buffer memories are mapped to the same addresses as the corresponding image buffer used by the 8096. Normally, the frame buffers are mapped to the CRT controller to keep the video monitor updated. However, when the image stored in the image buffer is changed, the 8096 performs a frame-synchronized flyby block move to refresh the frame buffer (50 to 290 ms depending on whether frame synchronization is off or on).

To digitize an image, the 8096 monitors the end-of-line and end-of-frame signals from the CID camera and synchronizes the A/D conversion of the pixel data to a pixel valid signal from the camera. The analog output signal of the camera ranges from 0 to 1 V, corresponding to the gray level intensity at each pixel. This 1 V range is amplified to a 5 V range before being input to one of the eight A/D inputs of the 8096.

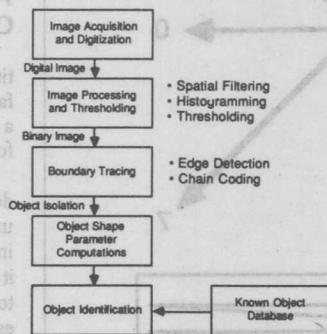


Figure 3. Object identification.

The 8096 converts the input voltage to a 10-bit digital representation in 22 μ s. Another 18 μ s are needed to store the pixel in the image buffer, update pointers, update a counter, and start another conversion. Therefore, the camera is clocked at a rate which results in a new pixel being output every 50 μ s.

Although the 8096 converts its analog input to 10 bits, the externally generated analog errors (such as buffer error and noise) led to the decision to use only 8 bits

of the result. This provides 256 gray levels, and greatly simplifies memory requirements.

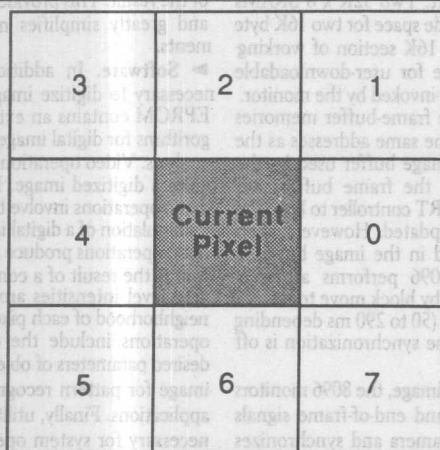
Software. In addition to the code necessary to digitize images, the system EPROM contains an extensive set of algorithms for digital image acquisition and analysis. Video operations are used to acquire a digitized image. Point and arithmetic operations involve the pixel-by-pixel manipulation of a digital image. Neighborhood operations produce an output image that is the result of a combination of the gray level intensities around a specified neighborhood of each pixel. Measurement operations include the computation of desired parameters of objects located in an image for pattern recognition and other applications. Finally, utility operations are necessary for system operation.

The algorithms present in the system monitor can be used to identify desired objects in a digital image by following the approach shown in Figure 3. Once an image is digitized, it can be enhanced by the application of various image processing techniques including histogramming, thresholding, and spatial filtering to delineate the desired objects. The 8-directional chain code (Figure 4) can then be used to trace the boundaries of objects, and relevant object parameters can be determined and compared with those of a known object database to identify the unknown object.

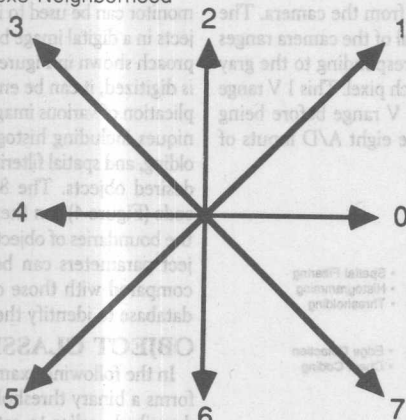
OBJECT CLASSIFICATION

In the following example, the 8096 performs a binary thresholding operation as described earlier to set the image background to pure white and the objects in the image to pure black. Then the 8096 searches the image for objects. When an object is found, the object boundary is traced and shape analysis is performed. Descriptive information about the object (or objects) is output over the on-chip UART of the 8096 to a terminal, or host computer. The controller, without consulting a host computer, can also be programmed to make the decision to accept or reject an object on a set of prescribed rules.

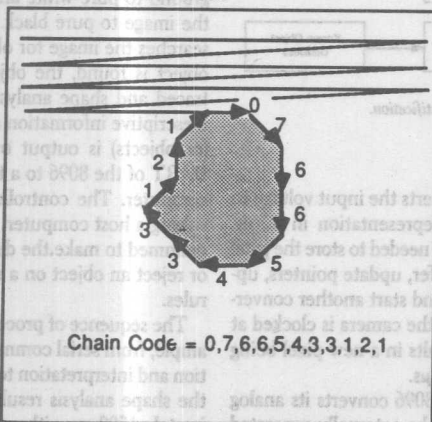
The sequence of processing for this example, from serial communication reception and interpretation to the reporting of the shape analysis results, takes approximately 1500 ms with an 8096 running at 12 MHz. The time will vary with the size and number of objects in the field of view.



(a) A 3x3 Neighborhood



(b) 8-Directional Chain-Code



(c) Example Chain-Code of an Object

Figure 4. 8-directional boundary chain code.

Photos 1 through 4 show the original 256 gray level digitized image and resultant binary (two-level) image of a circular object and a square object. (The circle looks like an oval when displayed due to the aspect ratio of the video monitor).

Table 1 summarizes the output of the systems shape analysis program. The objects' perimeter (P), area (A), center of mass coordinates (cx, cy), and the coordinates of the endpoints of each minimum enclosing rectangle are listed.

The rectangularity and circularity of the objects were also calculated and appear in Table 1. The rectangularity of the circle and the square using the actual data were ideal. Although the circularity of the digitized circle is slightly different from ideal (12.416 vs. 12.56), the digitized circle can be distinguished from the digitized square since the circularity of the square is very different from a perfect circle (15.44 vs. 12.56).

From these typical results, it is clear that this image processor can be used to distinguish between and identify objects placed in its field of view.

CONCLUSIONS

If the stringent requirements of "real-time" image processing can be relaxed in favor of substantially reduced system cost, a standard 16-bit microcontroller can perform as a stand-alone image processor.

Not only does the design described here demonstrate that a microcontroller can undertake two-dimensional image processing, but the surprising speed with which it accomplishes the processing should lead to the reevaluation of current microprocessor applications for possible cost reduction via microcontrollers.

REFERENCES

1. *Embedded Controller Handbook*, (latest edition). Intel Corporation, Santa Clara, CA.
2. Lin, S.H., A.L. Pai, and D.P. Ryan. *A Microcontroller Based Digital Image Processor*, Proceeding of the Second World Conference on Robotics Research, MS86-766, Society of Manufacturing Engineers, Dearborn, MI.
3. Cunningham, R. "Segmenting Binary Images," *Robotics Age*, Vol. 5, No. 2, July/August 1981.
4. Wilf, J.M. "Chain Code," *Robotics Age*, Vol. 3, No. 2, March/April 1981.
5. Gonzalez, R. and P. Wintz. *Digital Image Processing*, Second Edition, Addison-Wesley Publishing Company, NY, 1987.
6. Fu, S.U., R.C. Gonzalez, and C.S.C. Lee. *Robotics: Control, Sensing, Vision and Intelligence*, McGraw-Hill Book Company, NY 1987.

Image Processing Techniques

A histogram gives the distribution of all the gray levels in a digital image. The image histogram is used to select a desired threshold intensity level for separating an object from the background in the digital image.

A digital image can be thresholded using various threshold functions (three of which are shown in the figure) to yield an output image that contains a better definition of an object. For example, a binary (black and white) image is obtained by applying the two-level threshold function shown in (c).

Photo 1. A digitized image of a circle.

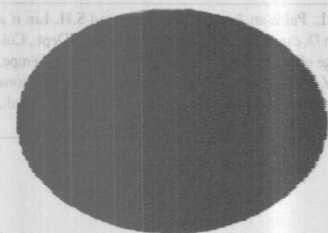


Photo 2. A thresholded binary (two-level) image of the same circle. The circle appears oval because of the aspect ratio of the video monitor.

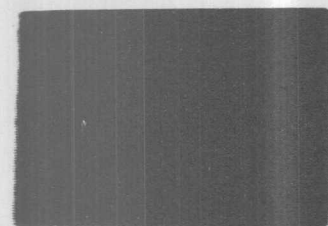


Photo 3. A digitized image of a square.

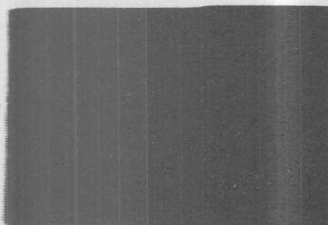
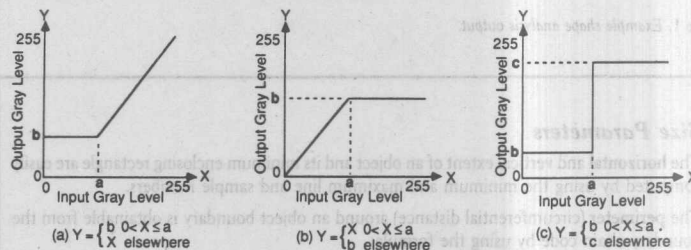


Photo 4. A thresholded binary image of the same square.



In spatial filtering, the pixels adjacent to pixel (x,y) of image plane f are operated upon by the filter mask operation h . The resulting value of this spatial convolution is used to compute a replacement gray level intensity value at location (x,y) in the output image g . The following formula is used:

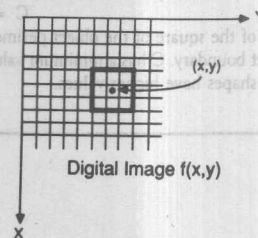
$$g(x,y) = h[f(x,y)] = [w_1 f(x-1,y-1) + w_2 f(x-1,y) + w_3 f(x-1,y+1) + w_4 f(x,y-1) + w_5 f(x,y) + w_6 f(x,y+1) + w_7 f(x+1,y-1) + w_8 f(x+1,y) + w_9 f(x+1,y+1)]$$

Various types of filter masks can be used to perform different digital image enhancement operations. A low-pass filter uses neighborhood averaging to "smooth" the digital image to remove noisy pixels. A high-pass filter accentuates noisy pixels. A high-pass filter accentuates the higher frequencies present in an image, thus "sharpening" its edges. Operators such as the Sobel masks can be used to compute the gradient at each point in an image, thus producing a gradient edge-detected image.

Using such filtering methods, the boundaries of objects in an image can be isolated, thus permitting the computation of useful object parameters for object identification and classification.

w_1 $(x-1,y-1)$	w_2 $(x-1,y)$	w_3 $(x-1,y+1)$
w_4 $(x,y-1)$	w_5 (x,y)	w_6 $(x,y+1)$
w_7 $(x+1,y-1)$	w_8 $(x+1,y)$	w_9 $(x+1,y+1)$

(a) A 3 x 3 Pixel Window with Spatial Mask coefficients w_i and Corresponding Image Pixel Locations



(b) A 3 x 3 Neighborhood Around Pixel (x,y)

$$\begin{bmatrix} 1/16 & 1/8 & 1/16 \\ 1/8 & 1/4 & 1/8 \\ 1/16 & 1/8 & 1/16 \end{bmatrix}$$

(c) A Low-Pass Filter Mask

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 9 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

(d) A High-Pass Filter Mask

$$\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \quad \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 2 & 1 \end{bmatrix}$$

(e) Sobel Gradient Masks

- Castleman, K.R. *Digital Image Processing*, Prentice Hall International, Englewood Cliffs, NJ, 1985.
- Baxes, G.A. *Digital Image Processing—A Practical Primer*, Prentice Hall International, Englewood Cliffs, NJ, 1984.

EXAMPLE SHAPE ANALYSIS OUTPUT

OBJECT	PERIMETER	AREA	C.O.M. COORDINATES	ENCLOSING RECTANGLE				RECTANGULARITY	CIRCULARITY
	P	A	CX,CY	XMAX	XMIN	YMAX	YMIN	$R=A_O/A_R$	$C=P^2/A$
CIRCLE	301	7297	(62,68)	111	15	116	21	0.785	12.416
SQUARE	328	6967	(55,73)	97	14	115	32	1.0	15.440

Table 1. Example shape analysis output.

Size Parameters

The horizontal and vertical extent of an object and its minimum enclosing rectangle are easily computed by using the minimum and maximum line and sample numbers.

The perimeter (circumferential distance) around an object boundary is obtainable from the boundary chain code by using the formula:

$$P = N_E + \sqrt{2} N_O$$

where N_E and N_O are the number of even and odd steps in the object boundary chain code.

The area of an object, which is a convenient measure of object size, is equal to the number of picture elements inside and including its boundary, multiplied by the area of a single pixel.

Shape Parameters

In addition to size parameters, shape parameters can be used to distinguish objects. Some shape parameters that are easily computed are described below.

The formula for computing the rectangularity R of an object is:

$$R = A_O / A_R$$

where A_O is the object area and A_R is the area of its minimum enclosing rectangle. R ranges from 0 to 1, with a value of 1.0 for rectangular objects, $\pi/4$ (0.785) for circular objects, and smaller values for slender, curved objects.

The aspect ratio, A , which is the ratio of width to length of the minimum enclosing rectangle of an object, is used to distinguish slender objects from roughly square or circular objects.

One of the commonly used circularity measures is:

$$C = P^2 / A$$

the ratio of the square of the object perimeter to its area, which reflects the complexity of the object boundary. C has a minimum value of 4π (12.56) for a circular object, while more complex shapes have higher values.

A.L. Pai is an Associate Professor and S.H. Lin is a Ph.D. candidate in the Computer Science Dept., College of Engineering, Arizona State University, Tempe, AZ 85281. David P. Ryan is a Senior Applications Engineer for Intel Corp., 5000 W. Chandler Blvd., Chandler, AZ 85226.

Reprinted from Sensors, June 1987
Copyright © 1987 by Helmers Publishing, Inc.
174 Concord St., Peterborough NH 03458
All Rights Reserved

SENSORS June 1987

MCS®-96 Diagnostic Library

7

MCS®-96 Diagnostic Library

7

1.0 INTRODUCTION

In the real time world of microcontroller applications, system failures can be dangerous, and expensive. Preventing them, and understanding them when they occur, is very important.

THE MCS®-96 DIAGNOSTICS LIBRARY

Version X1.1

David Ryan
INTEL Corporation
October 1987

General recovery from noise induced failures is possible using the WATCHDOG TIMER. While the 8096-based system is functional, the executing software periodically resets the WATCHDOG with a unique password which appears nowhere in the code map. This reduces the chance that an erroneous WATCHDOG reset would occur after a system upset.

The 8096 RESET instruction provides another form of protection. Since the opcode for a RESET is 00H, protection against the 8096 executing unimplemented external memory is obtained by placing pull-ups on the system bus. The RESET opcode is also the value in erased EPROMs. Therefore, any attempt to execute non-existent memory or an erased EPROM location causes the 8096 to execute the RESET instruction. RESET causes the 8096 to reinitialize itself and provides an external pulse on the RESET pin to reinitialize the system.

Even with the protection afforded by the 8096, a system is rarely complete without checks for hardware failures, both internal and external to the microcontroller. These checks are usually software routines that execute on power-up or periodically to verify that all parts of the system are present and function correctly. The tests generally execute standard check algorithms which are simply re-written in the host's assembly language.

To eliminate the need for every designer of an 8096-based system to write such tests, a collection of modular routines has been developed that any designer could easily use in his system (General Diagnostics). In addition, a set of 8096 interrupt service routines was developed for testing 8096 I/O units in a dedicated environment (The Dynamic Stability Test). Both sets of programs are contained in the MCS-96 Diagnostics Library (DIAG96.LIB).

This library is a collection of software modules that provide a number of ready-made General Diagnostics and a specialized MCS-96 diagnostic known as the Dynamic Stability Test. The General Diagnostics implement frequently used standard test algorithms, while the Dynamic Stability Test exercises hardware specific to the 8096.

The library can be considered a software "tool box" from which modules are selected for a variety of run-time diagnostics or laboratory tasks, for example:

- Include a few modules in other programs as a power-up test
- Use a memory module to create a map of external memory
- Use a few modules as a periodic system check
- Develop a simple stand-alone tester
- Build a custom test bed for burn-in, inspection or reliability tests

© Intel Corporation, 1987

October 1987
Order Number: 270083-002

1.0 INTRODUCTION

In the real time world of microcontroller applications, system failures can be dangerous, and expensive. Preventing them, and understanding them when they occur, is very important to the reliability of any design.

The sources of a system upset are varied. But in general, the failure of a well designed application occurs as a result of either some form of noise, or a hardware failure. The 8096 hardware provides methods of detecting and recovering from the transient noise failures, while the MCS[®]-96 Diagnostics Library supplies software routines that can help diagnose or detect a failure in system hardware.

Graceful recovery from noise induced failures is possible using the WATCHDOG TIMER. While the 8096-based system is functioning as desired, the executing software periodically resets the WATCHDOG with a special two-byte code. If the WATCHDOG is not reset at least every 16 ms (12 MHz system), a system reset occurs. The two-byte code is a unique password which appears nowhere in the opcode map. This reduces the chance that an erroneous WATCHDOG reset would occur after a system upset.

The 8096 RESET instruction provides another form of protection. Since the opcode for a RESET is 0ffH, protection against the 8096 executing unimplemented external memory is obtained by placing pull-ups on the system bus. The RESET opcode is also the value in erased EPROMs. Therefore, any attempt to execute non-existent memory or an erased EPROM location causes the 8096 to execute the RESET instruction. RESET causes the 8096 to reinitialize itself and provide an external pulse on the RESET pin to reinitialize the system.

Even with the protection afforded by the 8096, a system is rarely complete without checks for hardware failures, both internal and external to the microcontroller. These checks are usually software routines that execute on power-up or periodically to verify that all parts of the system are present and function correctly. The tests generally execute standard check algorithms which are simply re-written in the host's assembly language.

To eliminate the need for every designer of an 8096-based system to write such tests, a collection of modular routines has been developed that any designer could easily use in his system (**General Diagnostics**). In addition, a set of 8096 interrupt service routines was developed for testing 8096 I/O units in a dedicated environment (**The Dynamic Stability Test**). Both sets of programs are contained in the **MCS-96 Diagnostics Library (DIAG96.LIB)**.

This library is a collection of software modules that provide a number of ready-made **General Diagnostics** and a specialized MCS-96 diagnostic known as the **Dynamic Stability Test**. The **General Diagnostics** implement frequently used standard test algorithms, while the **Dynamic Stability Test** exercises hardware specific to the 8096.

The library can be considered a software "tool box" from which modules are selected for a variety of run-time diagnostics or laboratory tasks, for example:

- Include a few modules in other programs as a power-up test
- Use a memory module to create a map of external memory
- Use a few modules as a periodic system check
- Develop a simple stand-alone tester
- Build a custom test bed for burn-in, inspection or reliability tests
- Test new background code in an interrupt intensive environment

In addition to easing the development of a program that must perform standard diagnostics or system checks, the library can be a learning tool. Using the proven source code in the library, methods of interrupt management and on-chip peripheral handling can be reviewed to further understand how to use the 8096.

These tests were developed by the 8096 Applications group for experimental use with the 8096. With the programs in this library, the chip has been studied for its functional and asynchronous characteristics.

The **General Diagnostics** should be useful to almost anyone designing an 8096 application. The **Dynamic Stability Test** will be useful to those experimenting with the 8096 in a test environment. Figure 1 shows the modules in the **MCS-96 Diagnostics Library**.

1.1 General Diagnostics

The General Purpose Diagnostics consist of 24 programs providing System, ALU and Memory tests. Each of the tests can be called independently, and none require special hardware or impose application limiting constraints.

Two Collected Test programs are also provided so that all tests may be called at once. A third Collected Test program executes a selection of **General Diagnostics** that might be reasonable to include in a typical system power-up.

Section 3 provides a detailed description of the **General Diagnostics**.

1.2 Dynamic Stability Test

The **Dynamic Stability Test** is an integrated set of 11 programs that provide the interrupt service routines necessary to run all forms of MCS-96 I/O concurrently while a user written main task is executing. Virtually all of the chip is made to run simultaneously, with the I/O units responding to asynchronous external events.

Unlike the **General Diagnostics**, the **Dynamic Stability Test** modules must all be linked together, and must run in a specific external environment.

Section 4 provides a detailed description of the **Dynamic Stability Test**.

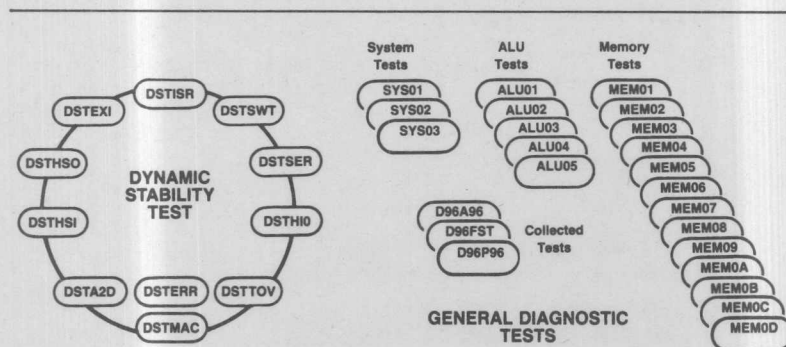


Figure 1. The MCS®-96 Diagnostic Library

1.3 How To Use This Manual

This publication is meant to be a guide for those using any of the programs in the MCS-96 Diagnostics Library. On a first pass the entire manual should be skimmed, with more attention paid to Section 2 and the overview sections of Sections 3 & 4. For the casual reader, the overview sections of each chapter should suffice.

Section 2 contains an overview of the general calling conventions to use any test in DIAG96.LIB. The section also describes DIAG96.LIB error reporting conventions and presents some warnings to heed when using this library.

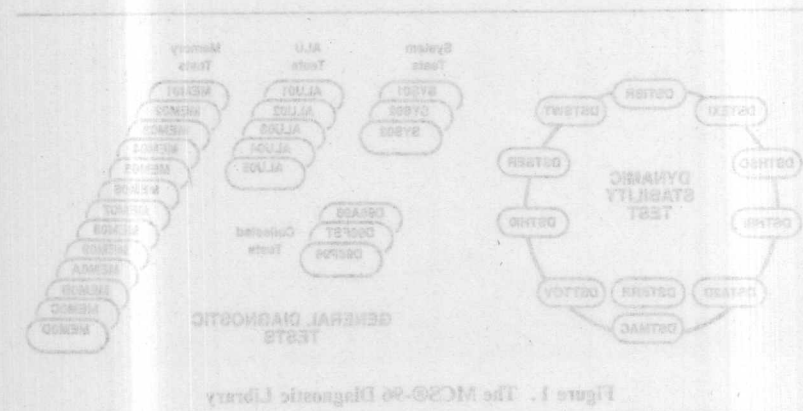
Section 3 describes the classes of **General Diagnostics** and each test in detail.

Section 4 describes the concept of Dynamic Stability and its implementation on the 8096. The section also contains an overview of the tests performed, a description of the constraints placed upon the user-written background task, and detailed descriptions of each interrupt service routine.

The Appendices contain error code and command file descriptions, and of demonstration program listings. Source for the **MCS-96 Diagnostics Library** can be obtained from Insite User's Program Library at the address below. The Insite Catalog order number is AE-17.

Insite User's Program Library
INTEL Corporation DV2-24
2402 W. Beardsley Road
Phoenix, Arizona 85027

With the first-hand knowledge that many problems result from not being able to uncover information lodged in some dark corner of the user manual, information is repeated in the sections where it is pertinent.



2.2 Using the General Diagnostics

2.0 USING THE LIBRARY

To simplify use of the diagnostics, the tests were developed in a modular fashion and collected in one linkable object file library (DIAG96.LIB). A modular program relies upon only the parameters sent at its invocation and employs standard parameter passing conventions to allow flexibility and uniformity of use. Collecting the modules into a library eliminates the tedium of listing twenty or thirty file names when performing a relocate/link on user developed code. When a program is linked to DIAG96.LIB, only those modules referenced in the user program are drawn from the library for inclusion in the output module.

Since PLM96 conventions were the ones chosen for this set of programs, the **General Diagnostics** are invoked by following the conventions for a PLM96 typed procedure. Parameters are placed on the STACK and the procedure activated via a function reference or explicit CALL. When the test is complete, test data is returned in the special register PLM\$REG. The **Dynamic Stability Test** is not PLM96 compatible.

The next section describes the format of the test data that is returned by the diagnostics. Following sections give an overview of how to use a **General Diagnostics** test, how to use **The Dynamic Stability Test**, and what restrictions to keep in mind while using the library.

2.1 Reporting Convention

All DIAG96.LIB tests use the PLM\$REG word locations 1CH and 1EH for returning condition codes to the calling program. Within DIAG96.LIB, these locations are the PUBLIC words EREG1 and EREG2. When a test concludes without finding an error, a zero is placed in the high byte of EREG1. If the high byte of EREG1 is non-zero, then some unexpected condition occurred. The low byte of EREG1 always contains the module number of the returning test, and EREG2 contains a detail code if an error was found. The complete listing of EREG1 code meanings and EREG2 meanings is in Appendices A & B.

All modules cease execution upon detection of the first error. The code describing which error was detected (EREG1) follows the format described in Table 1.

Table 1. Error Reporting Format

EREG1 = nnmx Hex			
where; nn = 00 if no error was found			
= 01, ..., 08H if an error was found, nn is the error code			
and;	mx	for Test =	SYS0x; 1 ≤ x ≤ 03H
			ALU0x; 1 ≤ x ≤ 05H
			MEM0x; 1 ≤ x ≤ 0DH
			D96A96; x = 0
			DSTISR; x = 0
			DSTHSI; x = 1
			DSTHSO; x = 0
			DSTHI0; x = 0
			DSTTOV; 0 ≤ x ≤ 1
			DSTEXI; x = 0
			DSTSER; x = 0
			DSTA2D; x = 0
			DSTSWT; 0 ≤ x ≤ 1
			D96FST; x = 0
			D96P96; x = 0

2.2 Using the General Diagnostics

The **General Diagnostics** provide a large set of system, ALU and memory tests that can be used in any combination, independent of system configuration or external circuitry. In addition to allowing for a wide flexibility in how a user's system is externally configured, the tests place minimal requirements on memory maps and interrupt environment.

Except where noted, all tests are interruptible, and maintain Program Status Word and Interrupt Mask integrity. The tests conform to PLM96 conventions, and require only run-time parameters to be passed for such specifics as memory test bounds and ALU test duration. To obtain access to the general diagnostics, the user should declare the needed module names **EXTERNAL** code segment symbols, and link to:

DIAG96.LIB

The tests are invoked in assembly language by placing the proper parameters on the **STACK** and **CALLing** the procedure. In PLM, the tests are run after a function reference is made with the appropriate parameters. The following is an example of an **ASM96** call to a memory test:

```
PUSH    #4000h
PUSH    #5000h
CALL    MEM06
CMPB    EREG1+1,0
BNE     Error_Found
```

The diagnostic module called performs a complementary address test on the byte locations between 4000H and 5000H inclusive. If an error is found, the value returned in the word **EREG1** will have a non-zero value as its high byte. Also in the case of an error, the **MEM06** memory test will place the address of the error in location **EREG2**. The program **D96A96**, shown in Appendix D is a working **ASM96** example that calls every **General Diagnostic Test**.

The same memory test could be called in a **PLM96** program as follows:

```
Response = MEM06(4000h,5000h);
IF Error$Codes.Number > 255 THEN CALL Error$Found;
```

Since the diagnostics return two words in the **PLM\$REG** locations **1CH** and **1EH**, the function **MEM06** would be a **PROCEDURE** of type **LONG**. **Error\$Codes** would have to be declared a **STRUCTURE AT Response**, with the word elements **Number** and **Detail** so that the error messages returned by the diagnostic can be stored. **Number** would contain the **EREG1** value returned by the test, and **Detail** would contain **EREG2**. **Response** would have to be **DECLARED** a double word. The program **D96P96**, shown in Appendix D is a working **PLM96** example that calls every **General Diagnostic**.

The action taken when an error is detected will depend upon the application. For example, the following **Error_Found** (or **Error\$Found**) routine would output the error codes to a printer or terminal:

```
Error_Found:      Error$Found: PROCEDURE;
PUSHF            DISABLE
PUSH    #Message_Ptr_A
CALL    Send_String
CALL output (.Message$Ptr$A,
            Error$Codes.Number);

PUSH    EREG1
CALL    Send_Hex_Word
CALL    Send_CR_LF
CALL output (.Message$Ptr$B,
            Error$Codes.Detail);

PUSH    #Message_Ptr_B
CALL    Send_String
Self: GOTO Self;
```

(Display continues on next page)


```

PUSH  EREG2
CALL  Send__Hex__Word
CALL  Send__CR__LF

```

```
BR $
```

```
Message_Ptr_A:
```

```
DCB 27,'ERROR FOUND. Error Number = '
```

```
Message_Ptr_B:
```

```
DCB 22,'Error Detail Code is = '
```

In the Error_Found routine, it is assumed that the subroutines Send_String, Send_Hex_Word, and Send_CR_LF transmit appropriate ASCII codes given the parameters passed to them. Send_String is sent a pointer to a byte string in memory, the first byte of which is the character count. Send_Hex_Word converts the word put on the STACK into the correct four ASCII code bytes and appends the ASCII code for H. Send_CR_LF outputs the ASCII codes to cause a carriage return, followed by a line feed. The PLM routine output would perform similar operations.

To run the test, the user must supply a background task that CALLS an initialization routine (DSTSR) with the specified parameters. After DSTSR returns, the interrupt service routines will begin running. The background task can then perform any function that conforms to the constraints discussed in Section 4. If the user does not wish to write a special background task, one is provided in the module DSTSR.

The following is an example CALL and a description of the parameters that must be passed to the initialization module (DSTSR).

```

CALL  DSTSR
PUSH  <bit pattern for memory test>
PUSH  <argument for MultiplyDivide Core test>
PUSH  <argument for MultiplyDivide Core test>
PUSH  <random test length>
PUSH  <random seed>
PUSH  <RAM segment ending address>
PUSH  <RAM segment starting address>
PUSH  <RAM segment ending address>
PUSH  <RAM segment starting address>

```

The RAM starting and ending addresses form a memory map for the memory tests that DSTSR runs. The internal RAM is always tested. The random seed is the starting point for ALU tests that execute for as many number pairs as is specified in the random test length parameter. Arguments and arguments are the operands for a MultiplyDivide test. The bit pattern parameter is used during a memory test of the internal RAM and the memory segments specified.

Section 4 contains more detailed information on using the Dynamic Stability Test, while the next section lists some general restrictions and assumptions that need to be understood to properly use any MCS-96 Diagnostic Library module.

2.3 Using the Dynamic Stability Test

The **Dynamic Stability Test** consists of a set of 8096 interrupt service routines that are designed to run while a user-supplied background task executes. The routines are located in the object file library DST96.LIB, which is contained in the master library DIAG96.LIB. To obtain access to the test, the user should invoke the batch file DSTRL.BAT with the background task file name and directory parameters. For example type:

```
DSTRL \SOURCE \BACK
```

Since the interrupt service routines test 8096 on-chip I/O devices, the part under test must reside in a specified hardware environment. Two such environments are available for use with the **Dynamic Stability Test**. The test may run in either a single chip mode, or a cross-coupled two chip mode. Figures 2 and 3 show the connections required for each configuration. In the single chip mode, output pins are connected to input pins on the same 8096. In the dual chip mode, output pins of one 8096 are connected to the input pins of the other (and vice versa).

To run the test, the user must supply a background task that CALLs an initialization routine (DSTISR) with the specified parameters. After DSTISR returns, the interrupt service routines will begin running. The background task can then perform any function that conforms to the constraints discussed in Section 4. If the user does not wish to write a special background task, one is provided in the module DSTUSR.

The following is an example CALL and a description of the parameters that must be passed to the initialization module (DSTISR).

```
PUSH    <RAM segment1 starting address>
PUSH    <RAM segment1 ending address>
PUSH    <RAM segment2 starting address>
PUSH    <RAM segment2 ending address>
PUSH    <random seed>
PUSH    <random test length>
PUSH    <argument1 for Multiply/Divide Core test>
PUSH    <argument2 for Multiply/Divide Core test>
PUSH    <bit pattern for memory test>
CALL    DSTISR
```

The RAM starting and ending addresses form a memory map for the memory tests that DSTISR runs. The internal RAM is always tested. The random seed is the starting point for ALU tests that execute for as many number pairs as is specified in the random test length parameter. Argument1 and argument2 are the operands for a Multiply/Divide test. The bit pattern parameter is used during a memory test of the internal RAM and the memory segments specified.

Section 4 contains more detailed information on using the **Dynamic Stability Test**, while the next section lists some general restrictions and assumptions that need to be understood to properly use any MCS-96 Diagnostic Library module.

2.4 Restrictions and Assumptions

Some general restrictions and assumptions need to be understood before any DIAG96.LIB programs can be successfully used.

- Pay close attention to the warnings about STACK location in the test modules you use. If you use any of the specialized internal register tests, make sure that the STACK is located externally. Do not partition a region of memory that contains your STACK in any memory test, unless you first move the STACK to an area you already tested.
- All General Diagnostics assume that the WATCHDOG TIMER is either being RESET by an interrupt service routine created by the user, or that it was never enabled. Only SYS02 ever locks out interrupts for a significant period of time. The amount of time they are locked out depends upon the parameters passed.
- The Dynamic Stability Test takes care of the WATCHDOG TIMER within its interrupt service routines. But, do not write to the WATCHDOG before CALLING the initialization subroutine.
- In any Dynamic Stability application, the user's Main Task should not lock out interrupts for more than a few instructions, as the CPU can get quite loaded down with interrupt requests that are very time dependent.

3.0 GENERAL DIAGNOSTICS

The 24 **General Diagnostics** included in DIAG96.LIB provide a good set of basic memory and ALU confidence tests that can be easily linked to application programs.

The **General Diagnostics** allow for a wide flexibility in how a user's system is configured with respect to memory maps and interrupt environment. Except where noted, all tests are interruptible, and maintain Program Status Word and interrupt mask integrity. The tests conform to PLM96 conventions, and require only run-time parameters to be passed for such specifics as memory test bounds and ALU test duration.

The tests are independent to allow specialized diagnostics to be developed as desired. Use just the quick power-up test (SYS02) to verify operation, or use the module that calls all **General Diagnostics** (D96A96) and let it run continuously for months. A module that performs the most common set of tests is also provided (D96FST).

The tests provided are of four classes: System Tests (SYSnn), ALU Tests (ALUnn), Memory Tests (MEMnn), and Collected Tests (D96xxx). To use any of the modules, from zero to ten parameters are PUSHed onto the STACK and the test is CALLED. Results are returned in the two word registers beginning at #1CH. The symbolic names for these locations (EREG1 and EREG2) are made PUBLIC if any DIAG96.LIB module is linked. They also may be referenced in PLM\$REG for PLM96 programs.

To obtain access to library modules, the user should declare the needed module names EXTERNAL code segment symbols, and link to:

DIAG96.LIB

The next few pages contain a brief overview of each of the four classes of tests. Then, the actions of each test are described in more detail.

System Tests

SYSnn

Common symbol definitions, storage reservations and two common routines are located in SYS01. A reference to any DIAG96.LIB module will cause SYS01 to be linked. SYS02 is meant to be called immediately after a RESET. It checks the special function register status and stack pointer, program status word and timer functionality. SYS03 is a simple program counter test. It does not test the complete range of the counter, and requires external RAM to execute.

SYS01: Common module

SYS02: RESET test

SYS03: Program counter exercise

ALU Tests

ALUnn

Five ALU modules are provided for checking ALU functionality. All report errors with a code in EREG1/EREG2.

Addition and subtraction are exercised in ALU01. A special eight-word add and subtract

is executed to test each adder bit with all possible combinations of a bit operation with and without carry-in.

Unsigned byte multiplication is verified by ALU02. This module simply executes all possible unsigned byte multiplications. Although not elegant, the test is effective. It takes six seconds.

A general test of the multiplication and division functions can be made with ALU03. The module executes all possible combinations of signed and unsigned, byte and word, two and three operand Multiplies and Divides using a specially selected table of numbers as operands.

ALU04 extends the ALU03 test by generating pseudo-random test pairs. The user program simply specifies a seed value for the random number generator, and the number of pairs to generate.

ALU05 is the core module for multiply/divide tests. Both ALU03 and ALU04 call ALU05. The user can also call ALU05 by passing a pair of test arguments. The module executes all possible combinations of signed and unsigned, byte and word, two and three operand Multiplies and Divides using the arguments passed as operands.

ALU01: Table-driven Addition/Subtraction
 ALU02: MULUB (all possible arguments)
 ALU03: Table-driven Multiply/Divide
 ALU04: Pseudo-random Multiply/Divide
 ALU05: Multiply/Divide core module

Memory Tests

MEMnn

The DIAG96.LIB MEMnn modules provide tests for register space, external RAM, and ROM. The algorithms used include: walking and galloping ones; walking and galloping zeros; checkerboard patterns; complementary addressing; and checksum verification.

The register tests are in MEM01-MEM05, and MEM0C. With the exception of MEM04, the register tests save the contents of all internal registers except PLM\$REG on the STACK before testing, and restore the data when done. If a faulty location is found, its address is reported. MEM04 is a utility which returns the number of bits set in a specified operand.

The external RAM tests are located in MEM06-MEM0A, and MEM0D. They all return a two-word code upon completion. The calling program must partition the RAM to be tested before calling an external RAM test.

Table 2. Memory Tests

Algorithm	Internal Registers	External RAM	ROM
Complementary Address	MEM01	MEM06	
Walking Ones		MEM07	
Walking Ones/Zeros	MEM02	MEM09	
Galloping Ones		MEM08	
Galloping Ones/Zeros	MEM03	MEM0A	
Bit Counter	MEM04		
Checkerboard Pattern	MEM05		
User Specified Pattern	MEM0C	MEM0D	
Checksum	MEM0B	MEM0B	MEM0B

Collected Tests

D96xxx

The D96xxx set of modules collects together all, or several, of the General Diagnostics and performs them according to the parameters passed. D96A96 is an ASM96 module that calls all tests. D96P96 is a PLM96 module that calls all tests. D96FST is an ASM96 module that calls a logical selection of tests.

D96A96: All tests / ASM96

D96P96: All tests / PLM96

D96FST: Selection of tests / ASM96

Memory Tests

MEM00

The DIAGN001B MEM00 modules provide tests for register space, external RAM, and ROM. The algorithms used include: walking and galloping ones; walking and galloping zeros; checksum patterns; complementary addressing; and checksum verification.

The register tests are in MEM01-MEM03, and MEM04. With the exception of MEM04, the register tests save the contents of all internal registers except PLMREG on the STACK before testing, and restore the data when done. If a faulty location is found, its address is reported. MEM04 is a utility which returns the number of bits set in a specified operand.

The external RAM tests are located in MEM06-MEM0A, and MEM0D. They all return a two-word code upon completion. The calling program must partition the RAM to be tested before calling an external RAM test.

Table 1. Memory Tests

Algorithm	Internal Registers	External RAM	ROM
Checksum	MEM0B	MEM0B	MEM0B
User Specified Pattern	MEM0C	MEM0D	
Checksum Pattern	MEM0B		
Bit Counter	MEM0B		
Galloping Ones/Zeros	MEM0B		
Galloping Ones	MEM0B		
Walking Ones/Zeros	MEM0B		
Walking Ones	MEM0B		
Complementary Address	MEM01	MEM0B	

3.1 System Tests

Common Symbols (SYS01)

Brief Description:

This module contains the global symbol declarations and five utilities used by the **General Diagnostics**.

Assembly Language Calling Sequence:

```

CALL    Get_Psw
or
CALL    Put_Psw
or
CALL    Get_Parms
or
CALL    Stack_Ram
or
CALL    Restore_Ram

Get_Psw Action:      Put_Psw Action:
USER_PSW := PSW      PSW := USER_PSW
EREG1 := 0
EREG2 := 0ffffh

Get_Parms Action:
PARM2 := Last Parameter
           put on the STACK
PARM1 := Next to last parameter
           put on the STACK
USER_PSW := PSW
EREG1 := 0ffh
EREG2 := 0000h

Stack_Ram Action:    Restore_Ram Action:
PUSH 1aH;             Ptr := 0feh;
Ptr := 20H;            Do While Ptr > 1eh;
Do While Ptr < 100h;    POP [Ptr];
    PUSH [Ptr] +        Ptr := Ptr-2;
End While;             End While;
                       POP 1aH;

```

Detailed Description:

A call to any **General Diagnostic** module will cause SYS01 to be linked. This module contains the definition of 4 words of memory used by every module to report errors and store temporary parameters. The STACK routines are used by the internal register tests to save and restore the data in the registers when called. It also INCLUDES an expanded 8096.INC file to provide the PUBLIC declarations of commonly used symbols for the special function registers and constants such as CR and LF.

Nearly all General Diagnostic modules use the routines in SYS01 to save the PSW when called, restore the PSW when returning control to the calling routine, save parameters from the STACK, and initialize the error registers.

System Power-up (SYS02)**Brief Description:**

This test is a quick check of the Program Status Word, TIMER1, IOS0, IOS1 and the Interrupt Pending Register. It is meant to be called just after a RESET.

Assembly Language Calling Sequence:

```
CALL    SYS02
```

When Test Passes:

EREG1 := 0002h

EREG2 := 0000h

If Test Fails:

EREG1 := 0102h on unexpected IOS0 or IOS1 — EREG2 := IOS0 in low byte
IOS1 in high byte

EREG1 := 0202h if TIMER1 does not change — EREG2 := TIMER1

EREG1 := 0302h if Zero register failed — EREG2 := PSW at Failure

EREG1 := 0402h if PUSHF/POPF failed — EREG2 := erroneous value found

EREG1 := 0502h if Sticky bit failed — EREG2 := 3fffh if bit did not set
:= 0000h if bit did not clear

EREG1 := 0602h if Carry Flag failed — EREG2 := xxxxh

EREG1 := 0702h on an overflow flag error — EREG2 := 0002h if flags set wrong
:= xxxxh flags cleared wrong

EREG1 := 0802h if Int. Pending byte failed — EREG2 := offending Int. Pend. value

Detailed Description:

This module verifies that TIMER1 is changing, then attempts to change the value in the ZERO register. Then, a set of PUSHFs and POPFs is done with test values to verify correct action of these instructions. The carry, sticky and overflow bits in the program status word are then tested. Finally, the Interrupt Pending register bits are tested for their ability to be set and cleared. Any unexpected result is reported.

Any error found having to do with the PUSHF/POPF instructions or the PSW, including Interrupt Pending, will cause interrupts to be disabled before returning to the calling module.

Program Counter (SYS03)**Brief Description:**

This test writes code into a user selected partition of RAM and executes the code. Elapsed time and special registers are checked for correctness.

Assembly Language Calling Sequence:

```
PUSH    <start address>
PUSH    <end address>
CALL    SYS03
```

When Test Passes:

EREG1 := 0003h

EREG2 := 0000h

If Test Fails:

EREG1 := 0103h if test code returned early

EREG2 := Early time

EREG1 := 0203h if test code returned late

EREG2 := Late time

EREG1 := 0303h if count register is incorrect

EREG2 := erroneous counter value

Detailed Description:

This module accepts starting and ending addresses for an external RAM partition, adjusts the boundaries to be double word aligned, and writes three lines of code repeatedly into the partition. The code that is written increments a counter then executes two NOPs every 12 state times. The last byte written into the RAM partition is a RET opcode.

After the RAM partition is adjusted and the code written into the RAM, the test puts a return address on the STACK, stores TIMER1 and CALLs the first byte of the RAM. When the last byte of RAM is executed, program control returns to SYS03. TIMER1 is again stored. The test then compares the elapsed time to the expected elapsed time. The value remaining in the counter is also checked for correctness. Any deviations from expected are reported.

Caution: Since interrupts are locked-out while the code in RAM is executing, partitioning more than 4000h bytes of RAM for this test could cause a WATCHDOG TIMER overflow if the watchdog was started before SYS03 is called.

3.2 ALU Tests

Add/Subtract (ALU01)

Brief Description:

This routine adds then subtracts two carefully selected eight-word variables and verifies the results.

Assembly Language Calling Sequence:

CALL ALU01

When Test Passes:

EREG1 := 0011h

EREG2 := 0000h

If Test Fails:

EREG1 := 0111h on an addition error

EREG1 := 0211h on a subtraction error

EREG1 := 0311h on a flag error

EREG2 := offending argument on error

Detailed Description:

Two eight-word operands are added together and the results verified. Then, the operands are subtracted and verified. The operands were chosen to exercise every possible combination of two bits and a carry into each bit of the adder. Correctness of the result and the resultant flags is verified.

The operands are:

```

05555555555555555555555555555555H
+ 05555555555555555555555555555555H
0AAB555500000000FFFE0000FFFFFFFFH
05555555555555555555555555555555H
- 0AAAA55555555555555555555555555H
0AAB555500000000FFFE0000FFFFFFFFH

```

Some versions of SIM96 do not pass this test.

MULUB (ALU02)**Brief Description:**

This module simply tests the MULUB instruction for all possible combinations of byte multipliers and multiplicands.

Assembly Language Calling Sequence:

```
CALL    ALU02
```

When Test Passes:

```
EREG1 := 0012h
```

```
EREG2 := 0000h
```

If Test Fails:

```
EREG1 := 0112h on an error
```

```
EREG2 := multiplier/multiplicand
```

Detailed Description:

This test executes all possible combinations of operands into the MULUB instruction. Results are verified through a method of addition and subtraction as operands cycle. The status of PSW flags is not verified in this routine.

Multiply/Divide Table (ALU03)**Brief Description:**

This module sends a specially constructed table of operands through the general Multiply/Divide Core test (ALU05).

Assembly Language Calling Sequence:

```
CALL    ALU03
```

When Test Passes:

```
EREG1 := 0013h
```

```
EREG2 := 0000h
```

If Test Fails:

```
EREG1 := 0115h on a signed error
```

```
:= 0215h on an unsigned error
```

```
:= 0315h on a flag error
```

```
EREG2 := offending argument on error
```

Detailed Description:

This test sends a table of operands through the Multiply/Divide Core test. The 18 operands were selected to exercise all of the hardware multiply and divide control signals.

The operands are:

Arg.1,Arg.2		Arg.1,Arg.2	
1D99H,	0FFFFH	0FFFFH,	9D99H
9D99H,	5555H	5555H,	0E266H
0E266H,	0AAAAH	0AAAAH,	1D99H
1D99H,	5555H	5555H,	9D99H
9D99H,	0AAAAH	0AAAAH,	0E266H
0E266H,	0FFFFH	0FFFFH,	0063H
0063H,	0055H	0055H,	0066H
0066H,	00AAH	00AAH,	0063H
0063H,	00FFH		

Some versions of SIM96 will not pass this test.

Multiply/Divide Random (ALU04)**Brief Description:**

This module is a pseudo-random number generator that sends pairs of arguments to the Multiply/Divide Core test (ALU05).

Assembly Language Calling Sequence:

```
PUSH    <seed>
PUSH    <count>
CALL    ALU04
```

When Test Passes:

EREG1 := 0014h
EREG2 := 0000h

If Test Fails:

EREG1 := 0115h on a signed error
EREG1 := 0215h on an unsigned error
EREG1 := 0315h on a flag error
EREG2 := offending argument on error

Detailed Description:

This module first executes the table driven Multiply/Divide test (ALU03). Then, if passed, pseudo-random argument pairs are generated and fed into the generalized Multiply/Divide Test (ALU05). The parameters passed to ALU04 set the random number seed, and the duration of the test.

There is no restriction on the values passed to the test. However, it must be noted that all possible combinations of signed and unsigned, byte and word, two and three operand Multiply/Divides are done at least twice for each pair of arguments sent to ALU05. Each such test takes from 1 to 5 milliseconds depending upon the arguments. Therefore, if large values for the count parameter are selected, the test will be long. For example, 1000h as a count will take about 12 seconds, depending upon the seed. NOTE: Some versions of SIM96 will not pass this test.

The formula used to generate the number pairs is as follows:

$$X(n+1) = [(0101h + 0001h) * X(n) + 0001h] \text{ MOD } 0ffffh$$

where X(0) = seed

Multiply/Divide Core (ALU05)**Brief Description:**

This test performs a Divide/re-Multiply sequence for all possible combinations of two or three operand, signed or unsigned, byte or word operations using the arguments passed to it as operands. The results are verified.

Assembly Language Calling Sequence:

```
PUSH    <argument1>
PUSH    <argument2>
CALL    ALU05
```

When Test Passes:

```
EREG1 := 0015h
EREG2 := 0000h
```

If Test Fails:

```
EREG1 := 0115h on a signed error
EREG1 := 0215h on an unsigned error
EREG1 := 0315h on a flag error
EREG2 := offending argument on error
```

Detailed Description:

This module takes arguments from a calling program and performs upon them all possible combinations of byte or word, two or three operand, signed or unsigned multiplication and division. Argument2 is used to create the high and low words for a word Divide, and the low byte of Argument1 is used as the divisor in a byte Divide.

The test checks multiplication and division by first dividing one operand by the other, then multiplying the quotient by the divisor and adding the remainder. If the result is the original dividend, the operations were correct. However, the possibility of legitimate division overflows must also be considered.

The test first performs a division and checks flag status for correct indication of overflow conditions. If there has been an overflow, the dividend is right shifted by one, the expected result is updated, and the division is performed over. If a division by zero occurred, just the expected result is corrected and the test is continued.

After a division and overflow check/fixup is complete, a re-multiplication occurs and the result verified. Flag status is also verified. If the results are correct, the original operands are reloaded into the test operand registers and the next Divide/re-Multiply combination is begun.

All Divide/Multiply combinations are performed twice. Once with flags set upon entry, and once with flags clear upon entry.

CALLING ALU03 will run a specially selected table of operands through this test. CALLING ALU04 will run a pseudo-random string of operands through this test.

3.3 Memory Tests

Complementary Address (MEM01) (for registers)

Brief Description:

This module performs a complementary address test on the registers locations 1ah to 0ffh.

Assembly Language Calling Sequence:

```
CALL    MEM01
```

When Test Passes:

```
EREG1 := 0021h
EREG2 := 0000h
```

If Test Fails:

```
EREG1 := 0121h
EREG2 := address of the error
```

Detailed Description:

This module performs a simple address and integrity test on register locations 1ah-0ffh. The algorithm stores the value NOT(ADDRESS) in the location pointed to by ADDRESS for the range, then loops through memory again to verify the contents.

Caution: If the STACK is partially internal, the STACK POINTER must be pointing at least 260 bytes into external RAM at the time MEM01 is called. The STACK cannot be entirely internal. The arithmetic flags in the PSW are undefined after execution of MEM01.

Brief Description:

This module performs a Walking Ones and Zeros test on the internal registers 1ah-0ffh.

Assembly Language Calling Sequence:

CALL MEM02

When Test Passes:

EREG1 := 0022h

EREG1 := 0000h

If Test Fails:

EREG1 := 0122h

EREG2 := address of the error

Detailed Description:

This module performs a Walking Ones and Zeros test on the internal registers.

The Walking Ones memory test first loads zero in all locations to be tested. Then, ones are placed in the first byte of memory, followed by a verification of all locations. Next, the first location is zeroed and ones are loaded into the second location. All memory is again verified. This process continues until all locations have been loaded with ones.

The Walking Zeros memory test works exactly like Walking Ones, except that a zero is "walked" through memory filled with ones, instead of ones being walked through a memory filled with zeros.

Caution: If the STACK is partially internal, the STACK POINTER must be pointing at least 260 bytes into external RAM at the time MEM02 is called. The STACK cannot be entirely internal. The arithmetic flags in the PSW are undefined after execution of MEM02.

**Galloping Ones/Zeros (MEM03)
(for registers)****Brief Description:**

This module performs a Galloping Ones and Zeros test on the internal registers 1ah-Offh.

Assembly Language Calling Sequence:

```
CALL    MEM03
```

When Test Passes:

EREG1 := 0023h

EREG2 := 0000h

If Test Fails:

EREG1 := 0123h

EREG2 := address of the error

Detailed Description:

This module performs a Galloping Ones and Zeros test on internal registers.

The Galloping Ones algorithm tests memory by first loading zeros into all locations. Then ones are loaded into the first byte and all memory is verified. The verification is done by alternating reads to the first location and locations through all memory. Next, ones are placed in the second location without altering the first. Verification is again performed by alternating reads to the second location and the rest of memory. This process continues until all locations contain ones.

The Galloping Zeros test is similar to Galloping Ones, except that zeros slowly fill a memory filled with ones. In Galloping Ones, ones slowly fill a memory filled with zeros.

Caution: If the STACK is partially internal, the STACK POINTER must be pointing at least 260 bytes into external RAM at the time MEM03 is called. The STACK cannot be entirely internal. The arithmetic flags in the PSW are undefined after execution of MEM03.

Bits Set (MEM04)**Brief Description:**

This module returns the number of bits set in the parameter passed to the routine.

Assembly Language Calling Sequence:

```
PUSH    test_value
CALL    MEM04
```

When All Bits Zero:

EREG1 := 0024h

EREG2 := 0000h

When One or More Bits Set:

EREG1 := 0124h

EREG2 := number of bits set

Detailed Description:

This module returns the number of bits that are set in the low byte of the parameter passed to the test. Any addressing mode may be used to put a value on the STACK, but the parameter on the STACK is treated as an immediate value.

Checkerboard Pattern (MEM05) (for registers)

Brief Description:

This module performs a Checkerboard Pattern test on the internal registers 1ah-0ffh.

Assembly Language Calling Sequence:

```
CALL    MEM05
```

When Test Passes:

```
EREG1 := 0025h
```

```
EREG2 := 0000h
```

If Test Fails:

```
EREG1 := 0125h
```

```
EREG2 := address of the error
```

Detailed Description:

This module performs a checkerboard test on the internal registers. A checkerboard pattern of ones and zeros is written into the physical rows and columns of the 8096 register space. As the pattern is being written, it is repeatedly verified. After the entire pattern is in place, the memory is verified again, complemented, and re-verified.

Caution: If the STACK is partially internal, the STACK POINTER must be pointing at least 260 bytes into external RAM at the time MEM05 is called. The STACK cannot be entirely internal. The arithmetic flags in the PSW are undefined after execution of MEM05.

Complementary Address (MEM06)

Brief Description:

This module performs a complementary address test on the memory partitioned by user supplied pointers.

Assembly Language Calling Sequence:

```
PUSH    <start address>
PUSH    <end address>
CALL    MEM06
```

When Test Passes:

```
EREG1 := 0026h
```

```
EREG2 := 0000h
```

If Test Fails:

```
EREG1 := 0126h
```

```
EREG2 := offending address
```

Detailed Description:

This module performs a simple address and integrity test on RAM locations partitioned by the parameters passed. The algorithm stores the value NOT(ADDRESS) in the location pointed to by ADDRESS for the range, then loops through memory again to verify the contents.

Caution: Do not partition RAM that contains valid STACK elements.

Walking Ones (MEM07)**Brief Description:**

This module performs a Walking Ones Test on the memory partitioned by the user.

Assembly Language Calling Sequence:

```
PUSH    <start address>
PUSH    <end address>
CALL    MEM07
```

When Test Passes:

```
EREG1 := 0027h
EREG2 := 0000h
```

If Test Fails:

```
EREG1 := 0127h
EREG2 := offending address
```

Detailed Description:

This module performs a Walking Ones test on the memory partitioned by the calling program. The Walking Ones memory test first loads zero in all locations to be tested. Then, ones are placed in the first byte of memory, followed by a verification of all locations. Next, the first location is zeroed and ones are loaded into the second location. All memory is again verified. This process continues until all locations have been loaded with ones.

Caution: Do not partition RAM that holds valid elements of the STACK. And, execution time increases non-linearly with memory partition widths.

Galloping Ones (MEM08)**Brief Description:**

This module performs a Galloping Ones test on memory partitioned by the calling program.

Assembly Language Calling Sequence:

```
PUSH    <start address>
PUSH    <end address>
CALL    MEM08
```

When Test Passes:

```
EREG1 := 0028h
EREG2 := 0000h
```

If Test Fails:

```
EREG1 := 0128h
EREG2 := offending address
```

Detailed Description:

This module performs a Galloping Ones test on memory locations partitioned by the calling program.

The Galloping Ones algorithm tests memory by first loading zeros into all locations. Then ones are loaded into the first byte and all memory is verified. The verification is done by alternating reads to the first location and locations through all memory. Next, ones are placed in the second location without altering the first. Verification is again performed by alternating reads to the second location and the rest of memory. This process continues until all locations contain ones.

Caution: Do not partition locations that contain valid elements of the STACK. And, execution time increases non-linearly with memory partition widths.

Walking Ones/Zeros (MEM09)**Brief Description:**

This module performs a Walking Ones and Zeros test on the memory locations partitioned by the calling program.

Assembly Language Calling Sequence:

```
PUSH    <start address>
PUSH    <end address>
CALL    MEM09
```

When Test Passes:

```
EREG1 := 0029h
EREG2 := 0000h
```

If Test Fails:

```
EREG1 := 0129h
EREG2 := offending address
```

Detailed Description:

This module performs a Walking Ones and Zeros test on the memory partitioned by the calling program.

The Walking Ones memory test first loads zero in all locations to be tested. Then, ones are placed in the first byte of memory, followed by a verification of all locations. Next, the first location is zeroed and ones are loaded into the second location. All memory is again verified. This process continues until all locations have been loaded with ones.

The Walking Zeros memory test works exactly like Walking Ones, except that a zero is "walked" through memory filled with ones, instead of ones being walked through a memory filled with zeros.

Caution: Do not partition RAM that contains valid elements of the STACK. And, execution time increases non-linearly with memory partition widths.

Galloping Ones/Zeros (MEM0A)**Brief Description:**

This module performs a Galloping Ones and Zeros test on the memory locations partitioned by the calling program.

Assembly Language Calling Sequence:

```
PUSH    <starting address>
PUSH    <ending address>
CALL    MEM0A
```

When Test Passes:

EREG1 := 002Ah

EREG2 := 0000h

If Test Fails:

EREG1 := 012Ah

EREG2 := offending address

Detailed Description:

This module performs a Galloping Ones and Zeros test on memory partitioned by the calling program.

The Galloping Ones algorithm tests memory by first loading zeros into all locations. Then ones are loaded into the first byte and all memory is verified. The verification is done by alternating reads to the first location and locations through all memory. Next, ones are placed in the second location without altering the first. Verification is again performed by alternating reads to the second location and the rest of memory. This process continues until all locations contain ones.

The Galloping Zeros test is similar to Galloping Ones, except that zeros slowly fill a memory filled with ones. In Galloping Ones, ones slowly fill a memory filled with zeros.

Caution: Do not partition RAM that contains valid elements of the STACK. And, execution time increases non-linearly with memory partition widths.

Checksum (MEM0B)**Brief Description:**

This module calculates a 16 bit checksum for the memory partition specified by the calling program.

Assembly Language Calling Sequence:

```
PUSH    <starting address>
PUSH    <ending address>
CALL    MEM0B
```

Test Returns:

EREG1 := 012bh

EREG2 := 16-bit checksum

Detailed Description:

This module performs a 16-bit checksum on the region of memory partitioned by the calling program. RAM or ROM may be partitioned. The module is non-destructive to RAM.

User Pattern (MEMOC) (for registers)

Brief Description:

This module performs a Checkerboard Pattern test on the internal registers 1ah-0ffh with a user specified bit pattern.

Assembly Language Calling Sequence:

```
PUSH    <desired bit pattern>
CALL    MEMOC
```

When Test Passes:

```
EREG1 := 002Ch
EREG2 := 0000h
```

If Test Fails:
EREG1 := 012Ch
EREG2 := address of the error

Detailed Description:

This module performs a checkerboard test on the internal registers with the bit pattern specified by the calling program. The pattern is written into the physical rows and columns of the 8096 register space. As the pattern is being written, it is repeatedly verified. After the entire pattern is in place, the memory is verified again, complemented, and re-verified.

Caution: If the STACK is partially internal, the STACK POINTER must be pointing at least 260 bytes into external RAM at the time MEMOC is called. The STACK cannot be entirely internal. The arithmetic flags in the PSW are undefined after execution of MEMOC.

User Pattern (MEMOD)

Brief Description:

This module performs a Checkerboard Pattern test on a specified region of memory with a specified pattern of bits.

Assembly Language Calling Sequence:

```
PUSH    <starting address>
PUSH    <ending address>
PUSH    <bit pattern>
CALL    MEMOD
```

When Test Passes:

```
EREG1 := 002dh
EREG2 := 0000h
```

If Test Fails:

```
EREG1 := 012dh
EREG2 := offending address
```

Detailed Description:

This module performs a checkerboard test on a region of memory that is specified by the calling program using a bit pattern which is also specified. First, the pattern is written into memory. As the pattern is being written, it is repeatedly verified. After the entire pattern is in place, the memory is verified again, complemented, and re-verified.

Caution: Do not partition RAM that contains valid elements of the STACK.

3.4 Collected Tests Modules

ALL Tests in ASM96 (D96A96)

Brief Description:

This module causes every **General Diagnostics** test to execute.

Assembly Language Calling Sequence:

```

PUSH    <RAM segment1 starting address>
PUSH    <RAM segment1 ending address>
PUSH    <RAM segment2 starting address>
PUSH    <RAM segment2 ending address>
PUSH    <random seed>
PUSH    <random test length>
PUSH    <top of code address>
PUSH    <argument1 for Multiply/Divide Core test>
PUSH    <argument2 for Multiply/Divide Core test>
PUSH    <bit pattern for memory test>
CALL    D96A96

```

When Tests All Pass: When a Test Fails:

```

EREG1 := 0030h      EREG1 := test module error code
EREG2 := code checksum  EREG2 := test module detail code

```

Detailed Description:

This module calls all **General Diagnostics** using the parameters passed by the calling program. The parameters needed by the test for proper execution specify two areas of external RAM for memory tests, the ending address of code to be checksummed, the seed and length of the random ALU test, two specific arguments to do the Multiply/Divide Core test, and a bit pattern for memory tests.

Execution speed of this test is highly dependent upon the memory partitions and the length requested for the random ALU test. For example, partitioning 1k and 8k regions of memory, and calling for 1000h random ALU tests, the test takes 3 hours to complete. Testing smaller regions of memory (i.e. 1k and 1k) can reduce test time to a few minutes.

Caution: An external STACK must be used with this test, and it must be in a part of memory outside that partitioned during the CALL.

ALL Tests in PLM96 (D96P96)**Brief Description:**

This module causes every **General Diagnostics** test module to execute.

PLM96 Calling Sequence:

D96P96(RAM segment1 starting address,
 RAM segment1 ending address,
 RAM segment2 starting address,
 RAM segment2 ending address,
 random seed, random test length,
 top of code address,
 argument1 for Multiply/Divide Core test,
 argument2 for Multiply/Divide Core test,
 bit pattern for memory tests);

When All Tests Pass:

PLMREG := 00F0h
PLMREG + 2 := 16-bit checksum

When a Test Fails:

PLMREG := module error code
PLMREG + 2 := module detail code

Detailed Description:

This module calls all **General Diagnostics** using the parameters passed during invocation. The parameters needed by the test for proper execution specify two areas of external RAM for memory tests, the ending address of code to be checksummed, the seed and length of the random ALU test, two specific arguments to do the Multiply/Divide Core test, and a bit pattern for memory tests.

Execution speed of this test is highly dependent upon the memory partitions and the length requested for the random ALU test. For example, partitioning 1k and 8k regions of memory, and calling for 1000h random ALU tests, the test takes 3 hours to complete. Testing smaller regions of memory (i.e. 1k and 1k) can reduce test time to a few minutes.

In his program, the user will have to DECLARE D96P96 an external procedure of the LONG type, with its parameters declared SLOW. The EREG1 and EREG2 values reported by library modules are placed in the long-word location at PLM\$REG.

The DECLARations in D96P96 show how any one General Diagnostic Module could be called from a PLM96 program. Each needed module needs to be DECLARED an external procedure of the LONG type.

Caution: An external STACK must be used with this test, and it must be in a part of memory outside that partitioned during the CALL.

Selected Tests in ASM (D96FST)**Brief Description:**

This is an ASM module that invokes a selected set of **General Diagnostic** tests.

Assembly Language Calling Sequence:

```

PUSH    <RAM segment1 starting address>
PUSH    <RAM segment1 ending address>
PUSH    <RAM segment2 starting address>
PUSH    <RAM segment2 ending address>
PUSH    <random seed>
PUSH    <random test length>
PUSH    <top of code address>
PUSH    <argument1 for Multiply/Divide Core test>
PUSH    <argument2 for Multiply/Divide Core test>
PUSH    <bit pattern for memory test>
CALL    D96FST

```

When Tests All Pass:

EREG1 := 00E0h
EREG2 := code checksum

When a Test Fails:

EREG1 := test module error code
EREG2 := test module detail code

Detailed Description:

This module calls the Power-up and Program Counter tests then all ALU tests. Then, Complementary Address, Galloping Ones/Zeros and Checkerboard tests are run on the internal registers. Finally, Complementary Address and specified pattern tests are done on external memory and the program is checksummed.

The parameters needed by the test for proper execution specify two areas of external RAM for memory tests, the ending address of code to be checksummed, the seed and length of the random ALU test, two specific arguments to do the Multiply/Divide Core test, and a bit pattern for memory tests.

Execution speed of this test is highly dependent upon the memory partitions and the length requested for the random ALU test. For example, partitioning 1k and 8k regions of memory, and calling for 1000h random ALU tests, the test takes about 20 seconds to complete. Testing smaller regions of memory (i.e. 1k and 1k) can reduce test time further.

Caution: An external STACK must be used with this test, and it must be in a part of memory outside that partitioned during the CALL.

The **Dynamic Stability Test** is a set of interrupt service routines designed to run over a user's background task in either one stand alone 8097, or two 8097s that are cross-coupled. In the stand alone mode, the chip's output pins are hooked to its input pins. In the dual chip mode, each controller's output pins are tied to the input pins of the other. The minimum configuration for each mode are shown in Figures 2 and 3. See Figure 11 for the circuit diagram of a board that can be jumpered for either configuration.

What is Dynamic Stability?

A "Dynamic Stability" test was developed to enable testing of the 8097 in an asynchronous environment. In the one chip mode, HSO events are synchronized with the HSI

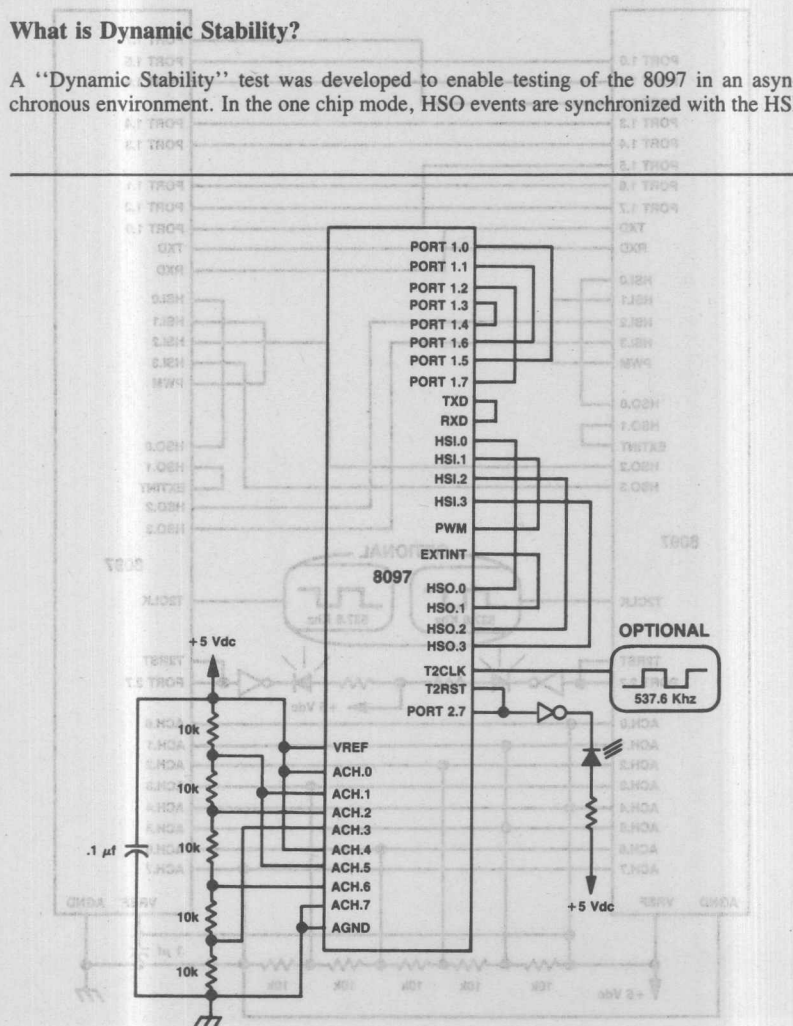


Figure 2. 8097 Strapback Configuration Single Chip Mode

event capture logic. However, in the cross-coupled mode, HSO events generated by one chip are captured in the HSI unit of another. As long as separate, non-synchronized clock sources are used for each chip, the HSI line events will occur asynchronously to the chip.

To implement a test that could be either stand alone or co-resident without modification, the creation and verification of I/O events needed to be decoupled. Thus the basic structure of the **Dynamic Stability Test** takes the form of a set of I/O Producers causing events that I/O Consumers verify. Figure 4 gives a macro view of the Producer/Consumer relationship.

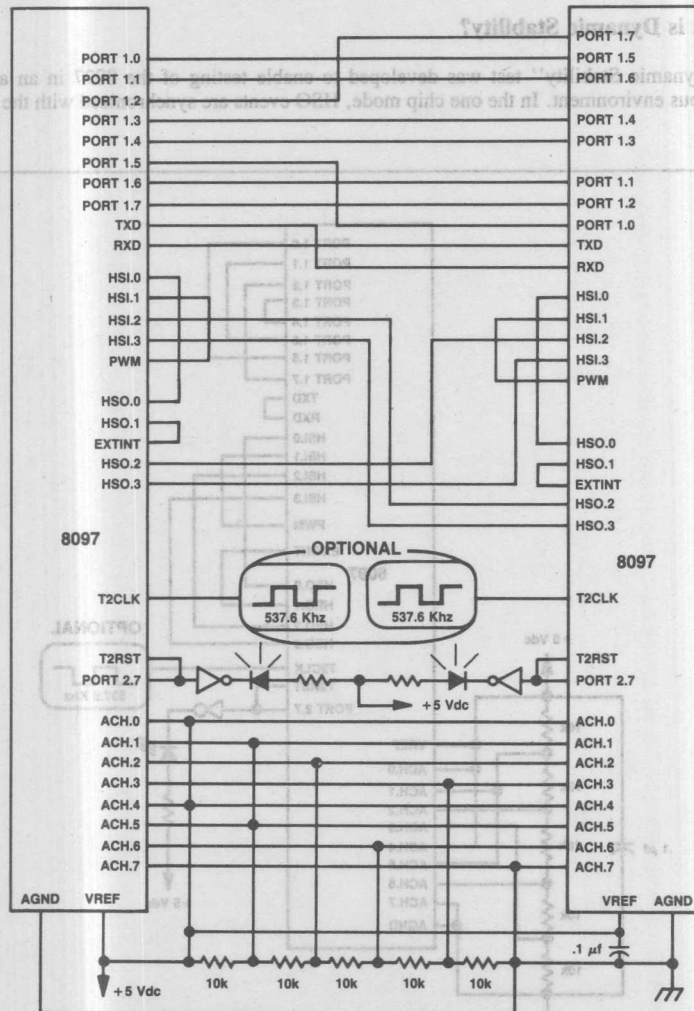


Figure 3. 8097 Strapback Configuration Dual Chip Mode

occur using this method is 9 state times ($2.25 \mu\text{s}$ in a 12 Mhz system). However, the skew should average between four and five state times. In any case, the parts will be far from synchronized shortly after the tests begin. This is fine, as long as the tests begin together.

In a one chip system, this process appears as a 600 microsecond pulse on PORT1.3. (See Figure 6.) The tests begin 600 microseconds after the rising edge.

When synchronization is complete, the interrupt service routines are initialized, interrupts are enabled, and control is returned to the background task. At this point, the testing really begins.

Producers and Consumers

The Producer/Consumer exchanges on the 8097 are executed by the interrupt service routines of the **Dynamic Stability Test**. While some interrupt routines contain an entire Producer or Consumer, some are spread through many routines. Figure 7 shows on a broad level the transactions that occur during test execution. Short descriptions of each Producer and Consumer follow, along with an indication of which interrupt routines contain them.

Serial Producer •DSTSER• The Serial Producer constantly transmits a table of alphabetic and special characters, and test data which includes the current status of the test and the REAL TIME since reset.

Serial Consumer •DSTSER• The Serial Consumer monitors the data coming over the serial link to see if all the expected characters are transmitted correctly and in the correct order. Transmission of the test data and the REAL TIME is checked by counting characters between carriage returns.

Port1 Producer •DSTSWT• The Port1 Producer outputs a series of values on Port1 that are contained in a table constructed to test all possible combinations of input and output of ones and zeros. The test producer executes every 5000h TIMER1 counts via the expiration of Software Timer 1.

Port1 Consumer •DSTSWT• The Port1 Consumer verifies the patterns appearing on Port1 using a table which contains the expected values. The check executes every 1000h TIMER1 counts via the expiration of Software Timer 2.

A/D Producer •DSTSWT• The A/D Producer continually starts A/D conversions by loading an HSO command to initiate an A/D. The A/D Producer executes every time Software Timer 0 expires.

A/D Consumer •DSTA2D• The A/D Consumer verifies the result of conversions initiated by the A/D Producer. It then changes the channel set for conversion and loads an HSO command to cause a Software Timer 0 expiration.

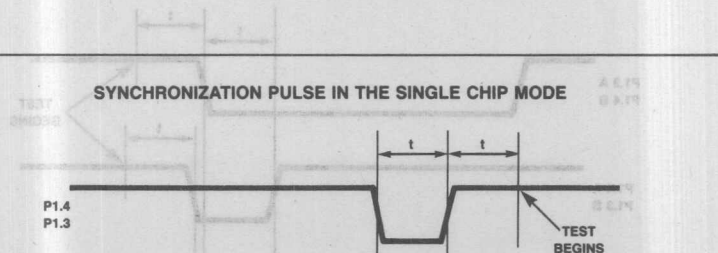


Figure 6. Single Chip Sync Pulse

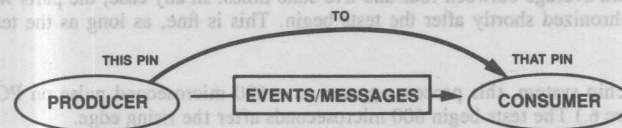


Figure 4. Producer/Consumer Relationship

What Does the Test Do?

Producer/Consumer exchanges were defined to test nearly all of the 8097 I/O capabilities concurrently. Following initialization, the transactions described are carried out by the set of interrupt service routines that make up the **Dynamic Stability Test**. The following section describes the test initialization. Then the tests performed are briefly described in the Producer/Consumer framework.

Initialization

To get the ball rolling, the background task must first CALL an initialization routine (DSTISR). This routine clears memory, executes the Selected Tests program (D96FST) from the **General Diagnostics**, and checks for the presence of an external clock on T2CLK. The serial port is then initialized for internal or external baud rate generation based on the presence of an external clock, and sign on messages are sent over the serial channel.

After initial tests are complete, and just prior to initiation of the interrupt service routines, a pulse is sent out on PORT1.3 that is used to synchronize controllers in the two chip mode. (See Figure 5.) Remember that the objective of the **Dynamic Stability Test** is to test the controllers asynchronously. Therefore, the synchronization is only done to insure that neither controller starts testing before both are ready to begin.

When a controller is ready to synchronize, it places a 0 on the PORT1.3 pin and looks for a 0 on its PORT1.4 pin. When a 0 is seen, the chip delays 600 microseconds, and then PORT1.3 is set high. The chip then loops until PORT1.4 also goes high. Another delay is inserted, and the tests begin. The worst skew between two controllers that can

SYNCHRONIZATION SEQUENCE IN THE DUAL CHIP MODE

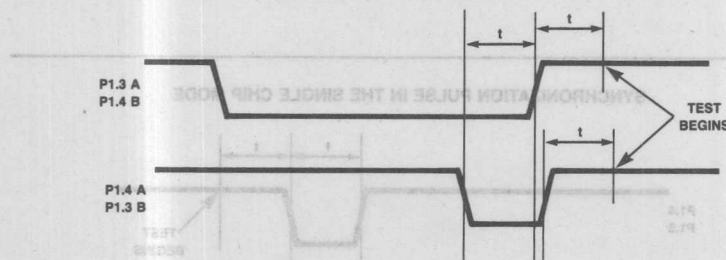


Figure 5. Dual Chip Synchronization

External Interrupt Producer •DSTHSO• The External Interrupt Producer causes rising edges on HSO.1, which is tied to EXTINT. This Producer executes every time there has been a falling edge on HSO.1.

External Interrupt Consumer •DSTEXI• The External Interrupt Consumer responds to rising edges on EXTINT. It resets the WATCHDOG TIMER every execution and tests the Test Status Words every 30h executions to see that all tests are running. This Consumer also loads an HSO command to cause a falling edge on HSO.1.

PWM Producer •DSTTOV• The PWM Producer executes every time there is a timer overflow. In addition to changing the PWM period, it toggles an LED and checks for unexpected T2CLK overflows. There is no PWM Consumer per se, but the PWM output is tied to HSI.1 which is configured to clock T2CLK. In this way T2CLK counts at a known average rate, and is used by the test in a modulo count fashion to generate a real

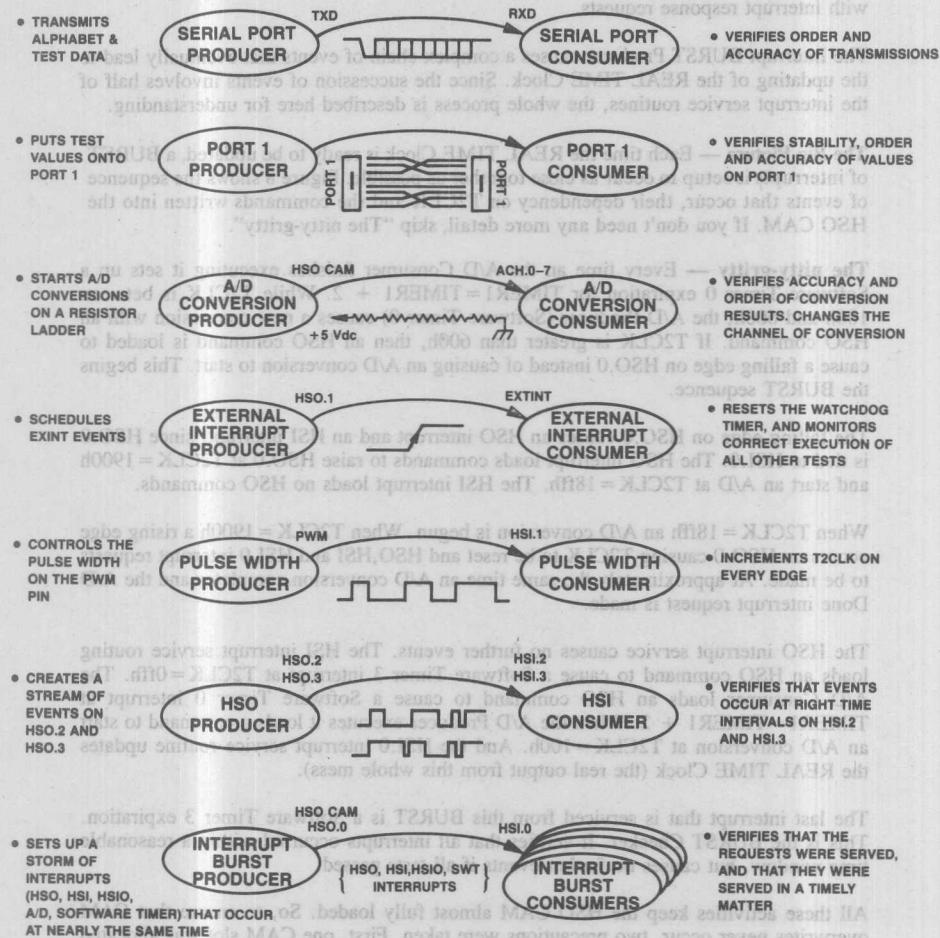


Figure 7. Producer/Consumer Overview

time clock. This module is also expandable to include tests that a user might want to execute only periodically.

HSO Producer •DSTHSO• The High Speed Output Producer executes every time an HSO event on HSO.2 or HSO.3 occurs. Varying pulse widths are created on the pins using predetermined tables of values. The minimum pulse width is 1000H; the maximum is 0C000H TIMER1 counts.

HSI Consumer •DSTHSI• The high speed inputs are monitored by the High Speed Input Consumer. The check executes every time an event occurs on HSI.2 or HSI.3. The HSI Consumer verifies that the proper pulse widths appear on the pins, and that the series of pulse widths is in the right order.

Interrupt BURST Producer •DSTSWT,DSTHI0,DSTHSO,DSTHSO• The previous Producer/Consumer transactions either go between controllers in the dual-chip mode, or stay within the same controller in the single-chip mode. However, there is one **Dynamic Stability Test** that executes invisibly to a co-controller in the dual-chip mode. This test, the Interrupt BURST Test, causes a flood of interrupts that almost fully load the 8097 with interrupt response requests.

The Interrupt BURST Producer causes a complex chain of events that eventually lead to the updating of the REAL TIME Clock. Since the succession of events involves half of the interrupt service routines, the whole process is described here for understanding.

The Big Picture — Each time the REAL TIME Clock is ready to be updated, a BURST of interrupts is setup to occur as close together as possible. Figure 8 shows the sequence of events that occur, their dependency on T2CLK and the commands written into the HSO CAM. If you don't need any more detail, skip "The nitty-gritty".

The nitty-gritty — Every time an the A/D Consumer finishes executing it sets up a Software Timer 0 expiration for $TIMER1 = TIMER1 + 2$. While T2CLK is between 100h and 600h, the A/D Producer (Software Timer 0) causes a new conversion with an HSO command. If T2CLK is greater than 600h, then an HSO command is loaded to cause a falling edge on HSO.0 instead of causing an A/D conversion to start. This begins the BURST sequence.

The falling edge on HSO.0 causes an HSO interrupt and an HSI interrupt, since HSO.0 is tied to HSI.0. The HSO interrupt loads commands to raise HSO.0 at $T2CLK = 1900h$ and start an A/D at $T2CLK = 18ffh$. The HSI interrupt loads no HSO commands.

When $T2CLK = 18ffh$ an A/D conversion is begun. When $T2CLK = 1900h$ a rising edge occurs on HSO.0 causing T2CLK to be reset and HSO,HSI and HSI.0 interrupt requests to be made. At approximately the same time an A/D conversion completes and the A/D Done interrupt request is made.

The HSO interrupt service causes no further events. The HSI interrupt service routing loads an HSO command to cause a Software Timer 3 interrupt at $T2CLK = 0ffh$. The A/D Consumer loads an HSO command to cause a Software Timer 0 interrupt at $TIMER1 = TIMER1 + 2$. When the A/D Producer executes it loads a command to start an A/D conversion at $T2CLK = 100h$. And the HSI.0 interrupt service routine updates the REAL TIME Clock (the real output from this whole mess).

The last interrupt that is serviced from this BURST is a Software Timer 3 expiration. This is the BURST Checker. It verifies that all interrupts occurred within a reasonable time window, but causes no further events if all tests passed.

All these activities keep the HSO CAM almost fully loaded. So, to ensure that CAM overwrites never occur, two precautions were taken. First, one CAM slot was allocated to four of the tests that use the HSO unit, and two slots were allocated for shared use by the Interrupt BURST process and the A/D conversion process.

The second precaution was to confirm that either the CAM was not full or the HOLDING REGISTER was empty (depending upon the test) before allowing any write to the CAM.

Figure 9 shows the HSO CAM loading over time, with T2CLK as the timebase. External Interrupt, Port1, HSO.2 and HSO.3 events each are allocated the use of one CAM slot all the time. While T2CLK is below 600h, but above 100h, another CAM slot is used by the A/D Done — Start A/D sequence. When T2CLK goes above 600h, two slots are used by the Interrupt BURST process. The BURST events conclude when T2CLK is reset and climbs to 100h. At 100h, the A/D Done — Start A/D sequence being again.

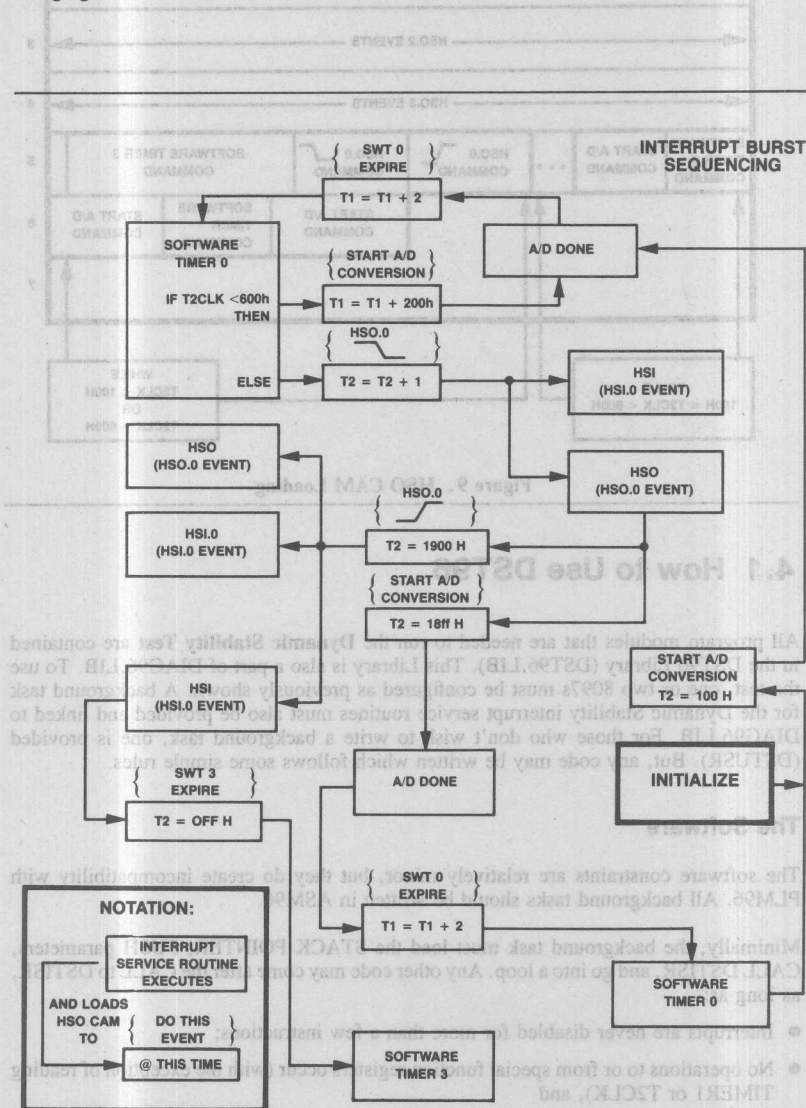


Figure 8. Interrupt BURST Sequencing

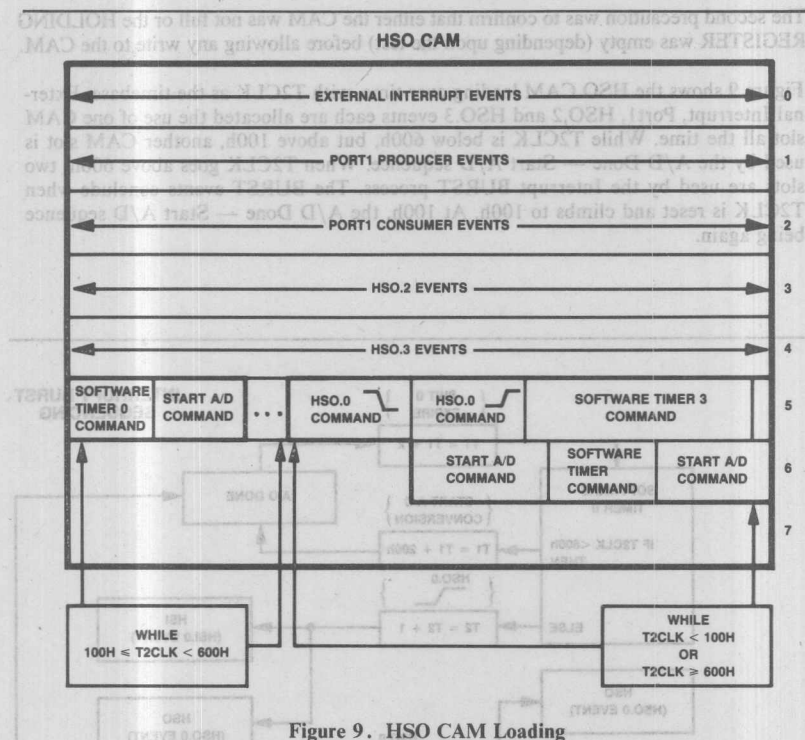


Figure 9. HSO CAM Loading

4.1 How to Use DST96

All program modules that are needed to run the **Dynamic Stability Test** are contained in the DST96 Library (DST96.LIB). This Library is also a part of DIAG96.LIB. To use the test, one or two 8097s must be configured as previously shown. A background task for the Dynamic Stability interrupt service routines must also be provided and linked to DIAG96.LIB. For those who don't wish to write a background task, one is provided (DSTUSR). But, any code may be written which follows some simple rules.

The Software

The software constraints are relatively minor, but they do create incompatibility with PLM96. All background tasks should be written in ASM96.

Minimally, the background task must load the STACK POINTER, PUSH parameters, CALL DSTISR, and go into a loop. Any other code may come after the CALL to DSTISR, as long as:

- Interrupts are never disabled for more than a few instructions;
- No operations to or from special function registers occur (with the exception of reading TIMER1 or T2CLK), and

Other less grave limitations on the main task are that it:

- Be CSEged at 2080h;

- Write only to EREG1, EREG2, OSEG registers from 40h to 5Ch, or external RAM, (the OSEG is an RL96 technicality, once DSTISR returns control to the MAIN TASK, locations 40h to 5Ch are not touched by the Tests); other registers can be read, but not written to;
- Communicate to the outside world through Port3 and Port4, (these Ports are untouched by the tests), or memory mapped I/O registers;

To provide the **Dynamic Stability Test** modules for linkage to your program, modify the batch file DSTRL.BAT to suit your system with respect to memory mapping and invoke the batch file with the appropriate background task filename. For example, type:

```
DSTRL DSTUSR
```

The Hardware

The **Dynamic Stability Test** has been designed to allow flexibility in the way output from the tests is used.

Minimally, no output device (printer, terminal) or function generators need to be attached to the test. If the LED attached to Port 2.7 is not flashing, the test failed. However, no other information may be gained.

To support a greater level of debugging (of the test code initially), the test was designed to output status and error information to one 4800 and one 300 baud device. The baud rates are derived from the function generators if present. Figure 10 shows how both devices can be attached to the test.

With this configuration, the test outputs an initialization message to both devices, then selects just the 4800 baud line for monitoring the Serial Port Producer/Consumer transactions. If an error is detected, the 300 baud line is selected for an error information dump.

A diagram of the circuit used in developing the **Dynamic Stability Test** appears in Figure 11. It is sufficiently general purpose for use in either the single or double chip modes, with or without printers or terminals attached.

The circuit requires that the 8097 I/O signals be present on an SBE-96 compatible 50 pin connector. The circuit also assumes that the analog voltage reference is provided through the cable. Therefore, if you are using the SBE-96, the jumpers to do this need to be in place (jumper numbers vary with the SBE-96 version).

Figure 12 describes how to jumper the **Dynamic Stability Test** board for one or two chip tests. Figure 13 shows the SBE-96 50 pin connector pinout. The following sections describe in detail the actions of each interrupt service routine in implementing the Producer/Consumer transactions.

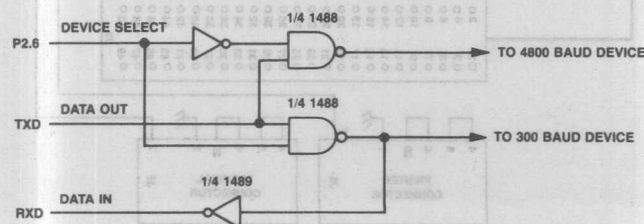
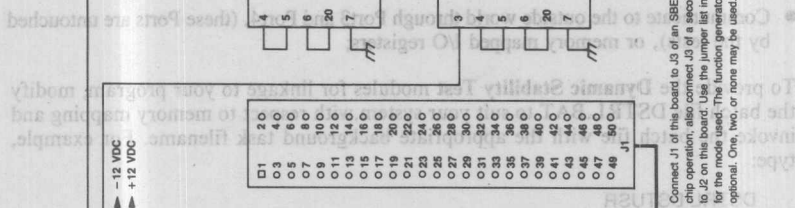


Figure 10. Output Device Selection Circuit



Jumper Connections for Single Chip Mode

J1		Also
22 - 37	34 - 28	
23 - 38	35 - 16	
24 - 39	42 - 31	
25 - 26	43 - 32	
45 - 48	46 - 29	E1 - E2
		E3 - E4
1 - 4 - 7 - 10 - 13		
15 - 18 - 21 - 27 - 30 - 33 - 36 - 41 - 44 - 47 - 50		

Jumper Connections for Dual Chip Mode

J1 - J2	J1 - J2	J1 - J2	J1	J2 - J1
22 - 37	33 - 33	14 - 14	34 - 28	22 - 37
23 - 38	36 - 36	4 - 4	45 - 48	23 - 38
24 - 39	41 - 41	5 - 5	46 - 29	24 - 39
25 - 26	44 - 44	7 - 7	35 - 16	25 - 26
42 - 31	47 - 47	10 - 10	J2	42 - 31
43 - 32	50 - 50	13 - 13		43 - 32
15 - 15	1 - 1		34 - 28	
18 - 18	9 - 9		45 - 48	
21 - 21	2 - 2		46 - 29	
27 - 27	8 - 8		35 - 16	
30 - 30	6 - 6			
	11 - 11		Also	
	3 - 3		E2 - E5	
	12 - 12		E1 - E4	

Figure 12. Dynamic Stability Board Jumper List

ANALOG GROUND	1	20	ANALOG CHANNEL 3
ANALOG CHANNEL 1	03	40	ANALOG GROUND
ANALOG CHANNEL 0	05	60	ANALOG CHANNEL 2
ANALOG GROUND	07	80	ANALOG CHANNEL 6
ANALOG CHANNEL 7	09	100	ANALOG GROUND
ANALOG CHANNEL 5	11	120	ANALOG CHANNEL 4
ANALOG GROUND	13	140	ANALOG VREF
DIGITAL GROUND	15	160	EXTERNAL INTERRUPT
RESET	17	180	DIGITAL GROUND
RXD	19	200	TXD
DIGITAL GROUND	21	220	PORT 1.0
PORT 1.1	23	240	PORT 1.2
PORT 1.3	25	260	PORT 1.4
DIGITAL GROUND	27	280	HSI.0
HSI.1	29	300	DIGITAL GROUND
HSO.4/HSI.2	31	320	HSO.5/HSI.3
DIGITAL GROUND	33	340	HSO.0
HSO.1	35	360	DIGITAL GROUND
PORT 1.5	37	380	PORT 1.6
PORT 1.7	39	400	PORT 2.6
DIGITAL GROUND	41	420	HSO.2
HSO.3	43	440	DIGITAL GROUND
PORT 2.7	45	460	PWM/PORT 2.5
DIGITAL GROUND	47	480	T2RST
T2CLK	49	500	DIGITAL GROUND

Figure 13. SBE-96 J3 Pinout

4.2 Test Module Descriptions

DST Initialization (DSTISR)

Brief Description:

This module is the invocation and initialization code for the Dynamic Stability Test.

Assembly Language Calling Sequence:

```

PUSH    <RAM segment1 starting address>
PUSH    <RAM segment1 ending address>
PUSH    <RAM segment2 starting address>
PUSH    <RAM segment2 ending address>
PUSH    <random seed>
PUSH    <random test length>
PUSH    <top of code address>
PUSH    <argument1 for Multiply/Divide Core test>
PUSH    <argument2 for Multiply/Divide Core test>
PUSH    <bit pattern for memory test>
CALL    DSTISR
  
```

When All Tests Pass:

```

EREG1 := 0040h
EREG2 := 0000h
  
```

When a Test Fails:

```

EREG1 := 0140h on abnormal RESET      EREG2 := TIMER1
EREG1 := 0240h if T2CLK won't change   EREG2 := xxxxh
EREG1 := 0340h if T2RST did not work    EREG2 := xxxxh
EREG1 := 0440h if IOC0.1 did not work   EREG2 := xxxxh
  
```

Detailed Description:

This module initializes the registers used by **Dynamic Stability Test** Modules, checks to see if there is an external clock present, tests T2CLK counting and reset functionality, and outputs initialization messages to the two output devices. The selected tests module (D96FST) from the **General Diagnostics** is also executed using the parameters specified.

When all initialization tests are passed, then a synchronization is performed to place the two processors in a dual-chip mode test in close sync. The PORT1 pins are used as to perform the handshaking synchronization. After synchronization, all **Dynamic Stability Tests** are activated and control is returned to the user program.

External Interrupts (DSTEXI)

Brief Description:

This module executes every time there is a rising edge on the EXTINT pin. The test resets the WATCHDOG TIMER and verifies execution of all Dynamic Stability routines.

If Test Fails:

EREG1 := 01A0h if a test did not execute

EREG2 := Number of Shifts done

Detailed Description:

This routine executes every time there is a rising edge on the EXTINT pin, causing an external interrupt. Each execution, the WATCHDOG TIMER is reset and an HSO command to clear the HSO.1 pin in 1000h TIMER1 counts is loaded into the CAM. The HSO routine that responds to that event will cause HSO.1 to go high, thus causing another vector to DSTEXI.

Every 30h executions of this module, the Test Status Words are NOTed and then NORMaLized to see if any test did not execute. If any bit in the Test Status Words is left set after being complemented, the NORML instruction will leave the most significant bit set, indicating an error. If there was no error, the TSWORDs are cleared. The user can change a mask in DSTEXI to enable checking of any of the currently spare bits in TSWORD. The TSWORD bit map is as follows:

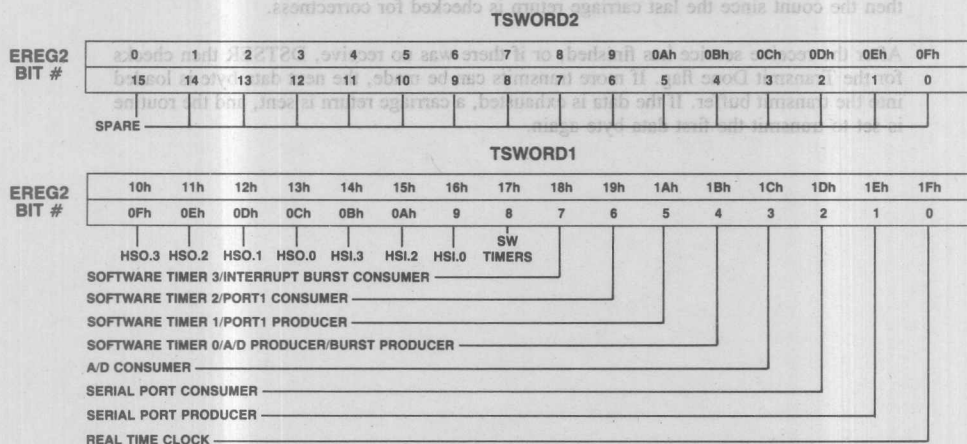


Figure 14. Test Status Word Bit Map

Serial Port (DSTSER)**Brief Description:**

This module contains the Serial Port Consumer and Producer routines for the **Dynamic Stability Tests**. It is executed on every Serial Interrupt.

If Test Fails:

EREG1 := 01B0h if a bad character was received
EREG2 := actual received character
EREG1 := 02B0h if an incorrect number of characters
 came between carriage returns
EREG2 := actual count

Detailed Description:

This interrupt service routine executes every time there is a Serial Interrupt. The data that is transmitted and checked by the test consists of first, the alphabet and some special characters; second, the current REAL TIME; and finally, the bit representation of the Test Status Words. The receiver verifies the alphabet and funny characters and counts characters until a carriage return. The following is an example of what the output looks like.

ABCDEFGHIJKLMNOPQRSTUVWXYZ*#%&[]@001:23:59.61 111111011101111110001111

The code first checks for a Receive Done flag. If a receive just completed, the receive buffer is emptied and checked for validity. If the received character is a carriage return, then the count since the last carriage return is checked for correctness.

After the receive service has finished, or if there was no receive, DSTSER then checks for the Transmit Done flag. If more transmits can be made, the next data byte is loaded into the transmit buffer. If the data is exhausted, a carriage return is sent, and the routine is set to transmit the first data byte again.

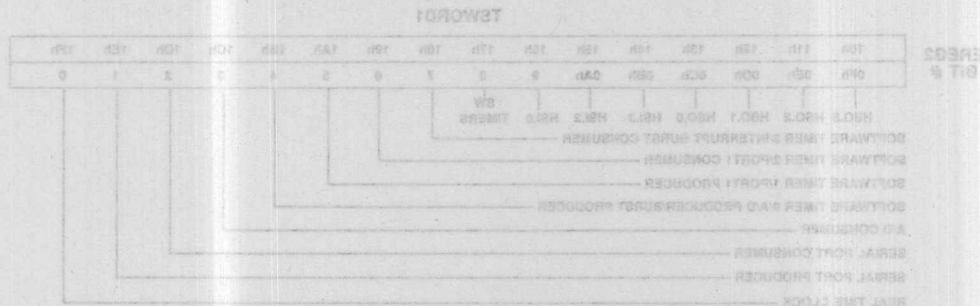


Figure 14. Test Status Word Bit Map

Software Timers (DSTSWT)**Brief Description:**

This module is executed every time a Software Timer Interrupt expires. The routine includes the Port1 Producer and Consumer, the A/D producer, and the Interrupt Burst control and verification code.

If a Test Fails:

EREG1 := 01D0h If an unexpected value is found on Port 1
EREG2 := expected value in high byte, actual value in low byte
EREG1 := 02D0h A/D Done interrupt did not occur within BURST window
EREG2 := Time between A/D done and Software Timer 0
EREG1 := 03D0h REAL TIME update did not occur within BURST window
EREG2 := Time between REAL TIME update and Software Timer 0
EREG1 := 04D0h HSO.0 response did not occur within BURST window
EREG2 := Time between HSO.0 interrupt and Software Timer 0
EREG1 := 05D0h HSI(.0) response did not occur within BURST window
EREG2 := Time between HSI(.0) service and Software Timer 0
EREG1 := 01D1h Invalid T2CLK value reached
EREG2 := T2CLK found
EREG1 := 02D1h Test reached an illegal Software Timer 0 state
EREG2 := the illegal case jump that was made

Detailed Description:

This module is called every time a Software Timer expires and causes an interrupt. Software timers are used by the A/D Done — A/D Trigger Sequence, the Interrupt Burst Sequence, and the Port1 Producer and Port1 Consumer.

When Software Timer 0 expires, a case jump is done on the BURST_STATE variable to sequence the A/D and interrupt BURST process to the appropriate state. Depending upon the value of T2CLK and the state of the A/D converter, either an A/D conversion is initiated or HSO.0 is set to go low to begin the interrupt BURST events.

When Software Timer1 expires, a new value is written to Port1 from a table constructed to test all combinations of input/output states on the quasi-bidirectional port pins. The HSO CAM is also loaded with a command to cause Software Timer1 to overflow again in 5000h TIMER1 counts.

When Software Timer 2 expires, Port1 is read and compared to a table of expected entries. If the value is correct, then an HSO command is loaded into the CAM to cause another Software Timer 2 expiration in 1000h TIMER1 counts. If the value is not correct, the next entry in the Table is checked. If there is still no match, an error is reported. If there is a match, the CAM loading occurs and Software Timer 3 is checked for expiration.

If Software Timer 3 has expired, then the flurry of BURST interrupts should have just occurred. The routine checks to see that each event happened within a reasonable time window. If the checks pass, then the routine exists with no further action.

Real Time Clock (HSI0) (DSTHI0)**Brief Description:**

This routine executes every time there is a rising edge on HSI.0 and updates the real time clock value.

When Module Executes:

REAL_TIME := REAL_TIME + .204 seconds

Detailed Description:

This module is the HSI.0 interrupt service routine. On each rising edge of HSI.0, the value in the REAL TIME clock buffer is updated to reflect the passing of 1900h T2CLK counts. Since the PWM output is tied to T2CLK, and the average time between edges is 31.875 μ s in a 12 MHz system, then 1900h T2CLK counts represents .204 seconds.

Execution of this module occurs during the interrupt BURST events. No action other than updating the REAL TIME clock is taken in this routine.

High Speed Outputs (DSTHSO)**Brief Description:**

This module manages the pulse width outputs on HSO.2 and HSO.3, and causes the Manager test to execute.

Detailed Description:

Every time an HSO command is executed that has the Interrupt bit set, this program executes. The routine manages the pulse widths on HSO lines two and three, and causes the Manager module to execute at the right time.

When a falling edge has been caused on either HSO.2 or HSO.3, DSTHSO loads a command into the CAM to cause a rising edge on the same line at a time that gives the line a low pulse width equal to a predetermined table value. Rising edges cause analogous responses. The tables used cause low and high pulse widths that vary from 1000h and 0C000h. The length of the tables differ by one so that all combinations of low and high table times occur.

When a falling edge was caused on HSO.1, the routine loads a command into the CAM to cause a rising edge on the same line two TIMER1 counts later. Since HSO.1 is tied to the EXTINT pin, rising edges cause the Manager Routine to execute.

Software Timer 3 expiration in 1000h TIMER1 counts. If the value is correct, the next entry in the Table is checked. If there is still no match, an error is reported. If there is a match, the CAM loading occurs and Software Timer 3 is checked for expiration.

If Software Timer 3 has expired, then the flurry of BURST interrupts should have just occurred. The routine checks to see that each event happened within a reasonable time window. If the checks pass, then the routine exits with no further action.

Brief Description:

This module does the verification of events on the HSI lines and initiates some interrupt BURST events when appropriate.

If a Test Fails:

EREG1 := 0161h if a high pulse on HSI.2 had an unexpected width
EREG2 := difference between actual and expected pulse width

EREG1 := 0261h if a low pulse on HSI.2 had an unexpected width
EREG2 := difference between actual and expected pulse width

EREG1 := 0361h if a high pulse on HSI.3 had an unexpected width
EREG2 := difference between actual and expected pulse width

EREG1 := 0461h if a low pulse on HSI.3 had an unexpected width
EREG2 := difference between actual and expected pulse width

EREG1 := 0561h if the HSI unit indicated that an HSI.1 event occurred
EREG2 := the time recorded in the FIFO

Detailed Description:

This module executes every time an event is loaded into the HSI Holding Register. Verification of pulse widths on HSI.2 and HSI.3 is done from tables of expected values. Any deviation is reported as an error.

If the test detects a negative transition on HSI.0, then commands are loaded into the HSO CAM to start an A/D at T2CLK = 18ffh and to set HSO.0 high at T2CLK = 1900h. This results in an HSO, HSI, HSI.0 and A/D Done interrupt requests to occur at approximately the same time — approaching a full demand on interrupt service.

When a rising edge on HSI.0 is detected, an HSO command is loaded into the CAM to cause a Software Timer 3 interrupt when T2CLK = 100h. The Software Timer 3 interrupt service will check to see that all burst events happened fast enough.

HSI.1 events are disabled from the FIFO. Any event detected on this line is reported as an error.

A/D Conversion Complete (DSTA2D)**Brief Description:**

This module executes every time an A/D conversion is complete. The conversion result is checked for correctness, the A/D converter is setup to convert on the next channel when initiated by an HSO command, and an HSO command to cause a Software Timer 0 expiration is loaded.

If Test Fails:

EREG1 := 01C0h on a conversion error

EREG2 := channel on which error occurred

Detailed Description:

This module executes every time an A/D conversion is complete. The conversion result is checked against a test table for correctness, and the A/D converter is setup to convert on the next channel when initiated by an HSO command. An HSO command to cause a Software Timer 0 expiration in 0002h TIMER1 counts is loaded just prior to exiting the module.

While T2CLK has a value between 100h and 600h, A/D conversions are initiated by the Software Timer 0 Interrupt service routine. When T2CLK goes above 600h, an A/D conversion is initiated by the HSO.0 interrupt service routine.

Given the possibility of additive error in 5% resistors, the conversion is tested to only six bits of accuracy.

Timer Overflows (DSTTOV)**Brief Description:**

This module toggles a port pin tied to an LED, manages the PWM output, performs some simple tests, and is expandable to allow inclusion of user written tests.

If Test Fails:

EREG1 := 0190h if T2CLK had an overflow indication

EREG2 := T2CLK at the time the error was found

Detailed Description:

This module executes every time TIMER1 or T2CLK overflow. Only TIMER1 overflows are valid however, so T2CLK overflows are flagged as an error. Each overflow, a new period is loaded into the PWMCONTROL register from a table of pulse periods. If an LED is connected, it will appear to slowly change in intensity. Port2.7 is also toggled in this routine to light another LED.

This interrupt routine can be expanded with special tests that are to execute on a periodic basis. Any of the spare bits in the Test Status Words can also be used by specialized tests. They will be checked by the External Interrupt service routine with a simple change in a bit mask.

Macro Module (DSTMAC)**Brief Description:**

This module contains four macros used by the **Dynamic Stability Test**.

Assembly Language Invocation:

```
SPSTATUS      Temp_Register
or
SPWAIT        (RI, TI)
or
BR_ON_ERROR   Label
or
RESET_WATCHDOG
```

Detailed Description:

The SPSTATUS Macro is used to ORB the Serial Port Status Register to a temp register. The Macro needs to be used to work around a bug in the 809x-90.

The SPWAIT Macro is used to cause program execution to halt and wait for an RI or TI flag, depending upon which is specified.

The BR_ON_ERROR Macro tests the high byte of EREG1 and jumps to the label if the byte is not zero. This can be used every time a **General Diagnostic** completes since the detection of any error will cause the high byte of EREG1 to be non-zero.

The RESET_WATCHDOG Macro does just what it says. The WATCHDOG TIMER is reset by writing the correct sequence to location 0Ah.

To access a DSTMAC macro, this module must be \$INCLUDED.

DST Example User Code (DSTUSR)**Brief Description:**

This is an example program that initiates the Dynamic Stability Test and then executes some General Diagnostics as a background task.

Detailed Description:

DSTUSR sends parameters defined at assembly time to the DST initialization routine (DSTISR). When control returns to DSTUSR, the example repeatedly executes ALU01, ALU02, ALU03, and MEM0A. It takes two minutes (with the given memory parameters) for the DSTUSR background task to cycle once while interrupts are running.

When creating a custom background task, using this example program as a template will speed development.

Brief Description:

This module is called if any error is detected in the **Dynamic Stability Test**. Information about the error is output over the serial port, and the test is restarted.

Assembly Language Calling Sequence:

CALL Error_Proc

Detailed Description:

This module is CALLED on detection of any error in the **Dynamic Stability Test**. When CALLED, the procedure:

- disables interrupts,
- saves any rapidly changing values (TIMER1, T2CLK, HSO_STATUS, ...),
- waits for a serial transmit in progress to complete,
- waits for the current serial receive to complete,
- empties eight entries from the HSI_FIFO,
- transmits an open loop sync sequence in case a co-controller is stuck in the sync routine, and
- waits a few hundred milliseconds to ensure that a co-controller has also detected a failure.

After these steps have been taken, the DSTERR de-selects the 4800 baud line, selects the 300 baud line, and outputs error messages. These messages include the Error Code (EREG1), the Detail Code (EREG2), the address of the line in the test which found the error, and the REAL TIME since reset.

Following the error messages, the procedure dumps the data contained in the registers and the external error buffer out over the serial port to the 300 baud device.

Finally, a RST instruction followed by a branch to the RST instruction is executed. If the WATCHDOG TIMER is externally disabled, the test will stay in this loop. If the WATCHDOG TIMER is not disabled, the test chip will reset, and the **Dynamic Stability Test** will reinitialize.

DST Example User Code (DSTUSR)

Brief Description:

This is an example program that initiates the **Dynamic Stability Test** and then executes some **General Diagnostics** as a background task.

Detailed Description:

DSTUSR sends parameters defined at assembly time to the DST initialization routine (DSTISR). When control returns to DSTUSR, the example repeatedly executes ALU01, ALU02, ALU04, ALU05 and MEM0A. It takes two minutes (with the given memory parameters) for the DSTUSR background task to cycle once while interrupts are running.

When creating a custom background task, using this example program as a template will speed development.

APPENDIX A

DIAG96.LIB Error Messages by EREG1 Code

APPENDICES

APPENDIX A • DIAG96.LIB Error Messages by EREG1 Code

APPENDIX B • DIAG96.LIB Error Messages by Module Name

APPENDIX C • Description of DIAG96.LIB Batch Files

APPENDIX D • Example Program Listings

D96A96

D96P96

D96FST

DSTUSR

APPENDIX A

DIAG96.LIB Error Messages by EREG1 Code

0000	No Message EREG2=0ffffh MODULE=SYS01/Common Symbols
0002	All Tests Passed EREG2=0000 MODULE=SYS02/System Power-up
0003	All Tests Passed EREG2=0000 MODULE=SYS03/Program Counter
0011	All Tests Passed EREG2=0000 MODULE=ALU01/Add/Subtract
0012	All Tests Passed EREG2=0000 MODULE=ALU02/MULUB
0013	All Tests Passed EREG2=0000 MODULE=ALU03/Multiply/Divide Table
0014	All Tests Passed EREG2=0000 MODULE=ALU04/Multiply/Divide Random
0015	All Tests Passed EREG2=0000 MODULE=ALU05/Multiply/Divide Core
0021	All Tests Passed EREG2=0000 MODULE=MEM01/Complementary Address (Registers)
0022	All Tests Passed EREG2=0000 MODULE=MEM02/Walking Ones/Zeros (Registers)
0023	All Tests Passed EREG2=0000 MODULE=MEM03/Galloping Ones/Zeros (Registers)
0024	No bits were set in the byte tested EREG2=0000 MODULE=MEM04/Bits Set
0025	All Tests Passed EREG2=0000 MODULE=MEM05/Checkerboard Pattern (Registers)
0026	All Tests Passed EREG2=0000 MODULE=MEM06/Complementary Address

MCS-96 Diagnostics Library

0027	All Tests Passed EREG2 = 0000 MODULE = MEM07/Walking Ones	A signed operation failed EREG2 = offending argument on error MODULE = ALU04/Multiply/Divide Table	0150
0028	All Tests Passed EREG2 = 0000 MODULE = MEM08/Galloping Ones	A signed operation failed EREG2 = offending argument on error MODULE = ALU05/Multiply/Divide Table	0150
0029	All Tests Passed EREG2 = 0000 MODULE = MEM09/Walking Ones/Zeros	A memory location failed EREG2 = address of the error MODULE = MEM09/Walking Ones/Zeros	0150
002A	All Tests Passed EREG2 = 0000 MODULE = MEM0A/Galloping Ones/Zeros	A memory location failed EREG2 = address of the error MODULE = MEM0A/Galloping Ones/Zeros	0150
002C	All Tests Passed EREG2 = 0000 MODULE = MEM0C/User Pattern (Registers)	A memory location failed EREG2 = address of the error MODULE = MEM0C/User Pattern (Registers)	0150
002D	All Tests Passed EREG2 = 0000 MODULE = MEM0D/User Pattern	A memory location failed EREG2 = address of the error MODULE = MEM0D/User Pattern	0150
0030	All Tests Passed, checksum is ready EREG2 = 16-bit checksum MODULE = D96A96/ALL Tests in ASM96	A memory location failed EREG2 = address of the error MODULE = D96A96/ALL Tests in ASM96	0150
0040	Initialization completed satisfactorily EREG2 = 0000 MODULE = DSTISR/DST Initialization	A memory location failed EREG2 = address of the error MODULE = DSTISR/DST Initialization	0150
00E0	All Tests Passed, checksum is over range specified EREG2 = 16-bit checksum MODULE = D96FST/Selected Tests in ASM	A memory location failed EREG2 = address of the error MODULE = D96FST/Selected Tests in ASM	0150
00F0	All Tests Passed, checksum is ready EREG2 = 16-bit checksum MODULE = D96P96/ALL Tests in PLM96	A memory location failed EREG2 = address of the error MODULE = D96P96/ALL Tests in PLM96	0150
0102	I/O Status Registers were unexpected EREG2 = I0S0 in low byte, I0S1 in high byte MODULE = SYS02/System Power-up	A memory location failed EREG2 = address of the error MODULE = SYS02/System Power-up	0150
0103	Test Code Returned Early EREG2 = Early Time MODULE = SYS03/Program Counter	A memory location failed EREG2 = address of the error MODULE = SYS03/Program Counter	0150
0111	An Addition error occurred EREG2 = offending argument when the error occurred MODULE = ALU01/Add/Subtract	A memory location failed EREG2 = address of the error MODULE = ALU01/Add/Subtract	0150
0112	Incorrect multiplication result was detected EREG2 = Multiplier/Multiplicand MODULE = ALU02/MULUB	A memory location failed EREG2 = address of the error MODULE = ALU02/MULUB	0150
0115	A signed operation failed EREG2 = offending argument on error MODULE = ALU03/Multiply/Divide Table	A memory location failed EREG2 = address of the error MODULE = ALU03/Multiply/Divide Table	0150

MCS®-96 Diagnostics Library

0115	A signed operation failed EREG2=offending argument on error MODULE=ALU04/Multiply/Divide Random	All Tests Passed EREG2=0000 MODULE=MEM01/Multiply/Divide Random	7500
0115	A signed operation failed EREG2=offending argument on error MODULE=ALU05/Multiply/Divide Core	All Tests Passed EREG2=0000 MODULE=MEM08/Multiply/Divide Core	8500
0121	A memory location failed EREG2=address of the error MODULE=MEM01/Complementary Address (Registers)	All Tests Passed EREG2=0000 MODULE=MEM01/Complementary Address (Registers)	9500
0122	A memory location failed EREG2=address of the error MODULE=MEM02/Walking Ones/Zeros (Registers)	All Tests Passed EREG2=0000 MODULE=MEM02/Walking Ones/Zeros (Registers)	0500
0123	A memory location failed EREG2=address of the error MODULE=MEM03/Galloping Ones/Zeros (Registers)	All Tests Passed EREG2=0000 MODULE=MEM03/Galloping Ones/Zeros (Registers)	0500
0124	At least one bit was set in the byte tested EREG2=number of bits set MODULE=MEM04/Bits Set	All Tests Passed EREG2=0000 MODULE=MEM04/Bits Set	0500
0125	A memory location failed EREG2=address of the error MODULE=MEM05/Checkerboard Pattern (Registers)	All Tests Passed, checksum is ready EREG2=16-bit checksum MODULE=MEM05/Checkerboard Pattern (Registers)	0600
0126	A memory location failed EREG2=address of error MODULE=MEM06/Complementary Address	Initialization completed satisfactorily EREG2=0000 MODULE=MEM06/Complementary Address	0600
0127	A memory location failed EREG2=address of the error MODULE=MEM07/Walking Ones	All Tests Passed, checksum is over EREG2=16-bit checksum MODULE=MEM07/Walking Ones	0600
0128	A memory location failed EREG2=address of the error MODULE=MEM08/Galloping Ones	All Tests Passed, checksum is ready EREG2=16-bit checksum MODULE=MEM08/Galloping Ones	0600
0129	A memory location failed EREG2=address of the error MODULE=MEM09/Walking Ones/Zeros	IO Status Registers were unexpected EREG2=1020 in low byte, 1021 MODULE=MEM09/Walking Ones/Zeros	0700
012A	A memory location failed EREG2=address of the error MODULE=MEM0A/Galloping Ones/Zeros	Test Code Returned Early EREG2=Early Time MODULE=MEM0A/Galloping Ones/Zeros	0700
012B	16-bit Checksum is ready EREG2=16-bit Checksum MODULE=MEM0B/Checksum	An addition error occurred EREG2=offending argument when MODULE=ALU01/Add/Subtract	0710
012C	A memory location failed EREG2=address of the error MODULE=MEM0C/User Pattern (Registers)	Incorrect multiplication result was detected EREG2=Multiply/Multiplicand MODULE=ALU02/Multiply	0710
012D	A memory location failed EREG2=address of the error MODULE=MEM0D/User Pattern	A signed operation failed EREG2=offending argument on MODULE=ALU03/Multiply/Divide	0710

MCS®-96 Diagnostics Library

0140	An abnormal RESET occurred EREG2=TIMER1 MODULE = DSTISR/DST Initialization	1850
0161	A high pulse on HSI.2 had an unexpected width EREG2= difference between actual and expected pulse width MODULE = DSTHSI/High Speed Inputs	0250
0190	An overflow of T2CLK was indicated EREG2=TIMER1 MODULE = DSTTOV/Timer Overflows	02D0
01A0	One or more DST Module did not execute on time EREG2= Number of SHIFTS done MODULE = DSTEXI/External Interrupt (Supervisor)	02D1
01B0	An unexpected serial character was received EREG2= Bad character received MODULE = DSTSER/Serial Port	0305
01C0	An unexpected A/D conversion result was found EREG2= Channel number of unexpected result MODULE = DSTA2D/A/D Conversion Complete	0305
01D0	Found unexpected value on PORT1 EREG2= expected value in high byte, actual in low byte MODULE = DSTSWT/Software Timers	0311
01D1	Invalid T2CLK value reached EREG2= T2CLK MODULE = DSTSWT/Software Timers	0315
0202	TIMER1 did not change over time EREG2= TIMER1 MODULE = SYS02/System Power-up	0315
0203	Test Code Returned Late EREG2= Late Time MODULE = SYS03/Program Counter	0315
0211	A Subtraction error occurred EREG2= offending argument when the error occurred MODULE = ALU01/Add/Subtract	0340
0215	An unsigned operation failed EREG2= offending argument on error MODULE = ALU03/Multiply/Divide Table	0351
0215	An unsigned operation failed EREG2= offending argument on error MODULE = ALU04/Multiply/Divide Random	0351
0215	An unsigned operation failed EREG2= offending argument on error MODULE = ALU05/Multiply/Divide Core	0351
0240	T2CLK will not change EREG2= xxxx MODULE = DSTISR/DST Initialization	

MCS®-96 Diagnostics Library

0261	A low pulse on HSI.2 had an unexpected width EREG2= difference between actual and expected pulse width MODULE= DSTHSI/High Speed Inputs	0140
02B0	A carriage return was received out of sequence EREG2= number of characters since a carriage return MODULE= DSTSER/Serial Port	0180
02D0	A/D Done did not occur within BURST window EREG2= Time between A/D done and Software Timer 0 MODULE= DSTSWT/Software Timers	0190
02D1	Test reached an illegal Software Timer 0 state EREG2= Illegal case jump made MODULE= DSTSWT/Software Timers	01A0
0302	Zero Register was found to change EREG2= Program Status Word At Failure MODULE= SYS02/System Power-up	01B0
0303	Counter Register contained unexpected value EREG2= Erroneous Counter Value MODULE= SYS03/Program Counter	01C0
0311	A flag error occurred EREG2= offending argument when the error occurred MODULE= ALU01/Add/Subtract	01D0
0315	A flag error occurred EREG2= offending argument on error MODULE= ALU03/Multiply/Divide Table	01D1
0315	A flag error occurred EREG2= offending argument on error MODULE= ALU04/Multiply/Divide Random	0200
0315	A flag error occurred EREG2= offending argument on error MODULE= ALU05/Multiply/Divide Core	0200
0340	T2RST pin would not RESET T2CLK EREG2= xxxx MODULE= DSTISR/DST Initialization	0211
0361	A high pulse on HSI.3 had an unexpected width EREG2= difference between actual and expected pulse width MODULE= DSTHSI/High Speed Inputs	0212
0391	Illegal Opcode	0212
03D0	REAL TIME update did not occur within BURST window EREG2= Time between REAL TIME update and Software Timer 0 MODULE= DSTSWT/Software Timers	0212
0402	PUSHF or POPF failed EREG2= Erroneous PUSHed or POPed value found MODULE= SYS02/System Power-up	0240

MCS®-96 Diagnostics Library

0440	I0C0.1 would not RESET T2CLK ERE2 = xxxx MODULE = DSTISR/DST Initialization
0461	A low pulse on HSI.3 had an unexpected width ERE2 = difference between actual and expected pulse width MODULE = DSTHSI/High Speed Inputs
04D0	HSO.0 response did not occur within BURST window ERE2 = Time between HSO.0 update and Software Timer 0 MODULE = DSTSWT/Software Timers
0502	Sticky Bit would not set ERE2 = 3fffh MODULE = SYS02/System Power-up
0502	Sticky Bit would not clear ERE2 = 0000 MODULE = SYS02/System Power-up
0561	HSI unit indicated an HSI.1 event occurred ERE2 = Time recorded in HSI FIFO MODULE = DSTHSI/High Speed Inputs
05D0	HSI(.0) response did not occur within BURST window ERE2 = Time between HSI(.0) service and Software Timer 0 MODULE = DSTSWT/Software Timers
0602	Carry Flag Test Failed ERE2 = xxxx MODULE = SYS02/System Power-up
0702	Overflow flags would not set correctly ERE2 = 0002h MODULE = SYS02/System Power-up
0702	Overflow flags would not clear correctly ERE2 = xxxx MODULE = SYS02/System Power-up
0802	Interrupt Pending Register failed read/write test ERE2 = offending Interrupt Pending byte MODULE = SYS02/System Power-up
xx91	(user defined) ERE2 = (user defined) MODULE = DSTTOV/Timer Overflows

APPENDIX B

DIAG96.LIB Error Messages by Module Name

ALU01	Add/Subtract	0011 All Tests Passed ERE2 = 0000
		0111 An Addition error occurred ERE2 = offending argument when the error occurred
		0211 A Subtraction error occurred ERE2 = offending argument when the error occurred
		0311 A flag error occurred ERE2 = offending argument when the error occurred
ALU02	MULUB	0012 All Tests Passed ERE2 = 0000
		0112 Incorrect multiplication result was detected ERE2 = Multiplier/Multiplicand
ALU03	Multiply/Divide Table	0013 All Tests Passed ERE2 = 0000
		0115 A signed operation failed ERE2 = offending argument on error
		0215 An unsigned operation failed ERE2 = offending argument on error
		0315 A flag error occurred ERE2 = offending argument on error
ALU04	Multiply/Divide Random	0014 All Tests Passed ERE2 = 0000
		0115 A signed operation failed ERE2 = offending argument on error
		0215 An unsigned operation failed ERE2 = offending argument on error
		0315 A flag error occurred ERE2 = offending argument on error
ALU05	Multiply/Divide Core	0015 All Tests Passed ERE2 = 0000
		0115 A signed operation failed ERE2 = offending argument on error
		0215 An unsigned operation failed ERE2 = offending argument on error
		0315 A flag error occurred ERE2 = offending argument on error

D96A96	All Tests in ASM96 0030 All Tests Passed, checksum is ready ERE2 = 16-bit checksum
D96FST	Selected Tests in ASM 00E0 All Tests Passed, checksum is over range specified ERE2 = 16-bit checksum
D96P96	ALL Tests in PLM96 00F0 All Tests Passed, checksum is ready ERE2 = 16-bit checksum
DSTA2D	A/D Conversion Complete 01C0 An unexpected A/D conversion result was found ERE2 = Channel number of unexpected result
DSTEX1	External Interrupt (Supervisor) 01A0 One or more DST Module did not execute on time ERE2 = Number of SHIFTs done
DSTHSIT	High Speed Inputs 0161 A high pulse on HSI.2 had an unexpected width ERE2 = difference between actual and expected pulse width 0261 A low pulse on HSI.2 had an unexpected width ERE2 = difference between actual and expected pulse width 0361 A high pulse on HSI.3 had an unexpected width ERE2 = difference between actual and expected pulse width 0461 A low pulse on HSI.3 had an unexpected width ERE2 = difference between actual and expected pulse width 0561 HSI unit indicated an HSI.1 event occurred ERE2 = Time recorded in HSI FIFO
DSTISR	DST Initialization 0040 Initialization completed satisfactorily ERE2 = 0000 0140 An abnormal RESET occurred ERE2 = TIMER1 0240 T2CLK will not change ERE2 = xxxx 0340 T2RST pin would not RESET T2CLK ERE2 = xxxx 0440 IOC0.1 would not RESET T2CLK ERE2 = xxxx
DSTSER	Serial Port 01B0 An unexpected serial character was received ERE2 = Bad character received 02B0 A carriage return was received out of sequence ERE2 = number of characters since a carriage return

MCS®-96 Diagnostics Library

DSTSWT	Software Timers	01D0 Found unexpected value on PORT1 ERE2 = expected value in high byte, actual in low byte
		01D1 Invalid T2CLK value reached ERE2 = T2CLK
		02D0 A/D Done did not occur within BURST window ERE2 = Time between A/D done and Software Timer 0
		02D1 Test reached an illegal Software Timer 0 state ERE2 = Illegal case jump made
		03D0 REAL TIME update did not occur within BURST window ERE2 = Time between REAL TIME update and Software Timer 0
		04D0 HSO.0 response did not occur within BURST window ERE2 = Time between HSO.0 update and Software Timer 0
		05D0 HSI(.0) response did not occur within BURST window ERE2 = Time between HSI(.0) service and Software Timer 0
		0190 An overflow of T2CLK was indicated ERE2 = TIMER1
		xx91 (user defined) ERE2 = (user defined)
		0121 A memory location failed ERE2 = address of the error
MEM01	Complementary Address (Registers)	0021 All Tests Passed ERE2 = 0000
		0121 A memory location failed ERE2 = address of the error
MEM02	Walking Ones/Zeros (Registers)	0022 All Tests Passed ERE2 = 0000
		0122 A memory location failed ERE2 = address of the error
MEM03	Gallopings Ones/Zeros (Registers)	0023 All Tests Passed ERE2 = 0000
		0123 A memory location failed ERE2 = address of the error
MEM04	Bits Set	0024 No bits were set in the byte tested ERE2 = 0000
		0124 At least one bit was set in the byte tested ERE2 = number of bits set

MCS®-96 Diagnostics Library

MEM05	Checkerboard Pattern (Registers) 0025 All Tests Passed ERE2 = 0000 0125 A memory location failed ERE2 = address of the error	Common Symbols 0000 No Message ERE2 = 0000 System Power-up 0000 All Tests Passed ERE2 = 0000	YS01 YS02
MEM06	Complementary Address 0026 All Tests Passed ERE2 = 0000 0126 A memory location failed ERE2 = address of error	NO Status Registers were ERE2 = 1080 in low byte 0200 TIMER did not ERE2 = 0000	
MEM07	Walking Ones 0027 All Tests Passed ERE2 = 0000 0127 A memory location failed ERE2 = address of the error	0200 Zero Register was found to change ERE2 = Program Status ERE2 = 0000 0400 PUSHF or POPF failed ERE2 = Error ERE2 = 0000	
MEM08	Galloping Ones 0028 All Tests Passed ERE2 = 0000 0128 A memory location failed ERE2 = address of the error	0500 Sticky Bit would not set ERE2 = 0000 0600 Sticky Bit would not clear ERE2 = 0000 0800 Carry Flag ERE2 = 0000	
MEM09	Walking Ones/Zeros 0029 All Tests Passed ERE2 = 0000 0129 A memory location failed ERE2 = address of the error	0700 Overflow flag would not ERE2 = 0000 0700 Overflow flag would ERE2 = 0000	
MEM0A	Galloping Ones/Zeros 002A All Tests Passed ERE2 = 0000 012A A memory location failed ERE2 = address of the error	0800 Interrupt Pending Register ERE2 = offending interrupt ERE2 = 0000 Program Counter 0000 All Tests Passed ERE2 = 0000	YS03
MEM0B	Checksum 012B 16-bit Checksum is ready ERE2 = 16-bit Checksum	0100 Test Code Returned Early ERE2 = Error ERE2 = 0000	
MEM0C	User Pattern (Registers) 002C All Tests Passed ERE2 = 0000 012C A memory location failed ERE2 = address of the error	0200 Test Code Returned Late ERE2 = Late Time 0300 Counter Register contains ERE2 = Error ERE2 = 0000	
MEM0D	User Pattern 002D All Tests Passed ERE2 = 0000 012D A memory location failed ERE2 = address of the error		

MCS®-96 Diagnostics Library

SYS01	Common Symbols	Check/Board Pattern (Registers)	MEM08
	0000 No Message	0002 All Tests Passed	
	ERE2 = 0ffffh	ERE2 = 0000	
SYS02	System Power-up	0128 A memory location failed	
	0002 All Tests Passed	ERE2 = address of the	
	ERE2 = 0000h		
	0102 I/O Status Registers were unexpected	Complementary Address	MEM08
	ERE2 = IOS0 in low byte, IOS1 in high byte	0002 All Tests Passed	
	0202 TIMER1 did not change over time	0128 A memory location failed	
	ERE2 = TIMER1	ERE2 = address of the	
	0302 Zero Register was found to change	Waiting Ones	MEM07
	ERE2 = Program Status Word At Failure	0002 All Tests Passed	
	0402 PUSHF or POPF failed		
	ERE2 = Erroneous PUSHed or POPed value found	ERE2 = address of the error	
	0502 Sticky Bit would not set	Gallop/ing Ones	MEM08
	ERE2 = 3ffffh	0002 All Tests Passed	
	0502 Sticky Bit would not clear	ERE2 = 0000	
	ERE2 = 0000	0128 A memory location failed	
SYS03	0602 Carry Flag Test Failed	ERE2 = address of the	
	ERE2 = xxxx		
	0702 Overflow flags would not set correctly	Waiting Ones	MEM08
	ERE2 = 0002h	0002 All Tests Passed	
	ERE2 = 0000	0128 A memory location failed	
	0702 Overflow flags would not clear correctly	ERE2 = address of the	
	ERE2 = xxxx		
	0802 Interrupt Pending Register failed read/write test	Gallop/ing Ones	MEM08
	ERE2 = offending Interrupt Pending byte	0002 All Tests Passed	
	ERE2 = 0000		
SYS03	Program Counter	0128 A memory location failed	
	0003 All Tests Passed	ERE2 = address of the	
	ERE2 = 0000		
	0103 Test Code Returned Early	Checksum	MEM08
	ERE2 = Early Time	0128 16-bit Checksum is	
	ERE2 = 16-bit Checksum		
SYS03	0203 Test Code Returned Late	User Pattern (Registers)	MEM08
	ERE2 = Late Time	0002 All Tests Passed	
	0303 Counter Register contained unexpected value	0128 A memory location failed	
	ERE2 = Erroneous Counter Value	ERE2 = address of the error	
SYS03		User Pattern	MEM08
		0002 All Tests Passed	
		ERE2 = 0000	
		0128 A memory location failed	

APPENDIX C

DESCRIPTION OF DIAG96.LIB BATCH FILES

The batch files that come with the library will help speed the process of either linking to the library as is, or revising library programs to suit custom purposes.

Some batch files require a parameter that provides the extensionless name of a user definable variable file to be included in the action of the batch file.

All DIAG96.LIB batch files assume that both the source and destination files reside in the same directory. Given the size of the library, and the fact that all of the files will not fit on one floppy disk, the command files will need to be edited if the user's system is not equipped with a hard disk.

INSTAL.BAT — Used to install the library on a hard disk system. To install the library, create a directory called \DIAG96 under the main directory, insert disk 1 into drive a: and type:

a:Instal

DST360K.BAT & DST12MEG.BAT — CAUTION: THESE BATCH FILES WILL FORMAT AND DESTROY ALL INFORMATION ON THE FLOPPIES USED.

These command files were created to make the DIAG96.LIB disk set. DST360K was created for use with 360K floppy disks and requires three diskettes. DST12MEG was created for use with 1.2M disks and only needs two diskettes. The batch files will prompt you to change disks. MAKE SURE TO ENTER THE CORRECT DISK DRIVE WHEN INVOKING THESE BATCH FILES. ALSO MAKE SURE TO INCLUDE THE DRIVE ID IN THE COMMAND LINE. THESE BATCH FILES FIRST FORMAT THE DISK, AND WE ALL KNOW WHAT WHEN DOS DEFAULTS TO THE HARD DISK!!!!!!!!!!

For example:

DST12MEG a:

SCRUB.BAT — CAUTION: THIS FILE DELETES FILES USING WILDCARDS.

All Diagnostic Library related files are deleted for the \DIAG96 directory. SYS?? and MEM?? wildcards are used, so be forewarned. This batch file does not delete itself!!!! To invoke this batch file, type:

Scrub

D96ASM.BAT — Assembles all General Diagnostic modules including the PLM compilation of D96P96.P96. To invoke the batch file, get in the \DIAG96 directory and type:

D96ASM

DSTASM.BAT — Assembles all Dynamic Stability Test modules. To invoke the batch file, get in \DIAG96 directory and type:

DSTASM

D96LP.BAT — Copies all General Diagnostic list files to a printer. Invocation must include a device where the printer resides. For example:

D96LP lpt1

DSTLP.BAT — Copies all Dynamic Stability Test modules to a printer. Invocation must include a device where the printer resides. For example:

DSTLP.BAT lpt1

LPONLY.BAT — Executes D96LP.BAT and DSTLP.BAT. Invocation must include a device where the printer resides. For example:

LPONLY lpt1

MCS®-96 Diagnostics Library

D96LIB.BAT — Deletes the current DIAG96.LIB collection. Also creates a new library of the same name using the files resident in the \DIAG96 directory bearing the **General Diagnostics** names. The DST96.LIB is not altered, and is included in the new DIAG96.LIB. To invoke the batch file, get in the \DIAG96 directory and type:

D96LIB

DSTLIB.BAT — Deletes the current DST96.LIB collection. Also creates a new library of the same name using the files resident in the \DIAG96 directory bearing the **Dynamic Stability Test** names. Since DST96.LIB is included in DIAG96.LIB, DIAG96.LIB is recreated by an invocation of D96LIB.BAT. To invoke this batch file, get in the \DIAG96 directory and type:

DSTLIB

DSTRL.BAT — This batch file is of most interest to **Dynamic Stability Test** users. It links a specified main task to the library. This file makes assumptions about the hardware memory implementation that may not be correct. Therefore minor changes may need to be made to the DSTRL.BAT RL96 invocation statement. A file name without extension must be provided and that file must reside in the \DIAG96 directory. The batch file assumes that the extension of the object file to be linked to the library is .OBJ. For example:

DSTRL Example_task

BLASTP.BAT — This batch file assembles the specified input file, then executes D96ASM.BAT, DSTASM.BAT, LPONLY.BAT, DSTLIB.BAT, and DSTRL.BAT. Then, the listfile output of the user's assembly and the print file of the linkage are copied to the printer specified. The batch file assumes that the input file is in the \DIAG96 directory and has a .A96 extension. For example:

BLASTP Example_lpt1

BLASTN.BAT — This batch file executes all assemblies, compilations, and linkages executed in BLASTP.BAT, but no copies are sent to the printer. The batch file assumes that the input file is in the \DIAG96 directory and has a .A96 extension. For example:

BLASTN Example_task

REGEN.BAT — Used to regenerate the library when only one module has changed. Specify the module that has changed when invoking this batch file. For example:

REGEN ALU03

MAKPLM.BAT — Used to make an impostor PLM96.LIB. The library created is not a real PLM96.LIB, and will not work with PLM programs. However, it is what is needed to use DIAG96.LIB. To invoke this batch file, get in the \DIAG96 director and type:

MAKPLM

MAKBH.BAT — Used to modify the library to run in an 8X9XBH. The 8X9XBH fails a flag test because of the -90 bug relating to the Z flag on add and subtract with carry is inadvertently verified by a library test. To invoke this batch file, get in the \DIAG96 directory and type:

MAKBH

D96RL.BAT — A generalized command that links target modules to DIAG96.LIB. It is intended for used when only the **General Diagnostics** are being used. Provide the target object file name and the directory in which it resides. For example:

D96RL \SOURCE \Example_

SERIES-III MCS-96 MACRO ASSEMBLER, V1.0

SOURCE FILE: :F5:D96A96.A96

OBJECT FILE: :F5:D96A96.OBJ

CONTROLS SPECIFIED IN INVOCATION COMMAND: NOGEN DEBUG

ERR LOC OBJECT

LINE SOURCE STATEMENT

```

1 112 BR ON ERR      ERROR LOGNG
2 117 *****
3 118 ALL TESTS ASM96  MODULE STACKSIZE(20)
4 119 ***** 0030
5 121
6 122 in order to run this module, the STACK must be ALL external, and the
7 123 data ram partitioned for memory test must not include ANY of the STACK
8 124
9 125 To call this module
10 126
11 127 PUSH    #<RAM segment1 start address>
12 128 PUSH    #<RAM segment1 ending address>
13 129 PUSH    #<RAM segment2 start address>
14 130 PUSH    #<RAM segment2 ending address>
15 131 PUSH    #<random seed>
16 132 PUSH    #<number of cycles desired for random test>
17 133 PUSH    #<address of the last byte of rom>
18 134 PUSH    #<an argument for mul/div tests>
19 135 PUSH    #<a second argument for mul/div tests>
20 136 PUSH    #<a bit pattern for memory tests>
21 137 CALL    D96A96
22 138
23 139
24 140 Remember, this test will take a long time if large memory regions are
25 141 partitioned, or if a large number of cycles of random test numbers is
26 142 requested. For example, with 8Kbytes of Ram in each region the test
27 143 executes in 3 hours.
28 144
29 145 It is suggested that for large memory tests, that the complimentary
30 146 address test be done on the whole region at once. Then, the more
31 147 exhaustive tests done on each memory chip in the system.
32 148 *****
33 149
34 150 EXTRN sp, ereq1, ereq2
35 151
36 152 EXTRN sys01, sys02, sys03, alu01, alu02, alu03, alu04, alu05
37 153 EXTRN mem01, mem02, mem03, mem04, mem05, mem06, mem07, mem08
38 154 EXTRN mem09, mem0a, mem0b, mem0c, mem0d
39 155
40 156 PUBLIC D96A96
41 157 $object
42 158
43 159 $include (:f3:dstmac.inc) ;provides the macro BR ON Error
44 160 *****
45 161 ;DST Macros INCLUDE FILE ;*****
46 162 ;*****

```

=1

=1

=1

ERR LOC	OBJECT	LINE	SOURCE STATEMENT	
		104		
		105		
0000		106	D96A96:	
		107		
0000 EF0000	E	108	CALL sys02	;CALL the System Power Up Test
		109		
		110	BR_ON_ERR Error_Found	
		114		
000A EF0000	E	115	CALL alu01	;CALL the Add/Subtract test
		116		
		117	BR_ON_ERR Error_Found	
		121		
0014 EF0000	E	122	CALL alu02	;CALL the MULUB test
		123		
		124	BR_ON_ERR Error_Found	
		128		
001E EF0000	E	129	CALL alu03	;CALL the Multiply/Divide Table
		130		;driven test
		131	BR_ON_ERR Error_Found	
		135		
0028 CB000C	E	136	PUSH 0ch[sp]	;PUSH a random seed
002B CB000C	E	137	PUSH 0ch[sp]	;PUSH the number of tests desired
002E EF0000	E	138	CALL alu04	;CALL the Multiply/Divide Random test
		139		
		140	BR_ON_ERR Error_Found	
		144		
0038 CB0006	E	145	PUSH 06h[sp]	;PUSH an argument
003B CB0006	E	146	PUSH 06h[sp]	;PUSH another argument
003E EF0000	E	147	CALL alu05	;CALL the Multiply/Divide Core Test
		148		
		149	BR_ON_ERR Error_Found	
		153		
0048 EF0000	E	154	CALL mem01	;CALL a Complementary Address test
		155		;on the internal registers
		156	BR_ON_ERR Error_Found	
		160		
0052 EF0000	E	161	CALL mem02	;CALL a Walking 1s/0s test on
		162		;the internal registers
		163	BR_ON_ERR Error_Found	
		167		
005C EF0000	E	168	CALL mem03	;CALL a Galloping 1s/0s test on
		169		;the internal registers
		170	BR_ON_ERR Error_Found	
		174		
0066 CB00	E	175	PUSH zero	;PUSH a zero
0068 EF0000	E	176	CALL mem04	;CALL the Bits Set Test
		177		
		178	BR_ON_ERR Error_Found	
		182		
0072 EF0000	E	183	CALL mem05	;CALL a Checkerboard Pattern test
		184		;for internal registers
		185	BR_ON_ERR Error_Found	
		189		
007C CB0014	E	190	PUSH 14h[sp]	;PUSH the start address
007F CB0014	E	191	PUSH 14h[sp]	; and the end address of a RAM area
0082 EF0000	E	192	CALL mem06	; and CALL a Complementary Address test
		193		

ERR LOC	OBJECT	LINE	SOURCE STATEMENT	
0188		194	BR_ON_ERR Error_Found	
008C CB0010	E	198		
008F CB0010	E	199	PUSH 10h[sp]	; PUSH a second start and end address
0092 EF0000	E	200	PUSH 10h[sp]	; and repeat the
		201	CALL mem06p[ab]	; Complementary Address test
0138		202		
		203	BR_ON_ERR Error_Found	
009C CB0014	E	207		
009F CB0014	E	208	PUSH 14h[sp]	; PUSH a start address
00A2 EF0000	E	209	PUSH 14h[sp]	; PUSH an ending address
		210	CALL mem07p[ab]	; CALL a Walking Ones test
		211		
		212	BR_ON_ERR Error_Found	
00AC CB0010	E	216		
00AF CB0010	E	217	PUSH 10h[sp]	; PUSH the start and end address
00B2 EF0000	E	218	CALL mem07p[ab]	; for another section of RAM
		219	CALL mem07p[ab]	; and repeat the Walking Ones test
		220		
		221	BR_ON_ERR Error_Found	
00BC CB0014	E	225		
00BF CB0014	E	226	PUSH 14h[sp]	; PUSH a start address
00C2 EF0000	E	227	PUSH 14h[sp]	; PUSH an ending address
		228	CALL mem08p[ab]	; CALL a Galloping Ones test
		229		
		230	BR_ON_ERR Error_Found	
00CC CB0010	E	234		
00CF CB0010	E	235		
00D2 EF0000	E	236	PUSH 10h[sp]	; PUSH a second start and end address
		237	PUSH 10h[sp]	; for another region of RAM and
		238	CALL mem08p[ab]	; CALL the Galloping Ones test again
		239		
		240	BR_ON_ERR Error_Found	
00DC CB0014	E	244		
00DF CB0014	E	245	PUSH 14h[sp]	; PUSH the start and end address of
00E2 EF0000	E	246	PUSH 14h[sp]	; a region of RAM
		247	CALL mem09p[ab]	; CALL the Walking 1s/0s test
		248		
		249	BR_ON_ERR Error_Found	
00EC CB0010	E	253		
00EF CB0010	E	254		
00F2 EF0000	E	255	PUSH 10h[sp]	; PUSH the start and end address of
		256	PUSH 10h[sp]	; another region of RAM
		257	CALL mem09p[ab]	; CALL the Walking 1s/0s test again
		258		
		259	BR_ON_ERR Error_Found	
00FC CB0014	E	263		
00FF CB0014	E	264	PUSH 14h[sp]	; PUSH the start and end address of
0102 EF0000	E	265	PUSH 14h[sp]	; a region of RAM
		266	CALL mem0ap[ab]	; CALL a Galloping 1s/0s test
		267		
		268	BR_ON_ERR Error_Found	
010C CB0010	E	272		
010F CB0010	E	273		
0112 EF0000	E	274	PUSH 10h[sp]	; PUSH the start and end address of
		275	PUSH 10h[sp]	; another region of RAM
		276	CALL mem0ap[ab]	; CALL the Galloping 1s/0s test again

ERR LOC	OBJECT	LINE	SOURCE STATEMENT	
		278	BR_ON_ERR Error_Found	
		282	BR_ON_ERR Error_Found	
		283		
011A	CB0002	EE 284	PUSH 02h[sp]	;PUSH a bit pattern to use and
011D	EF0000	EE 285	CALL mem0c	;CALL the Checkerboard Pattern test
		EE 286		; for internal registers
		287	BR_ON_ERR Error_Found	
		291	BR_ON_ERR Error_Found	
0125	CB0014	E 292	PUSH 14h[sp]	;PUSH the start and end address
0128	CB0014	EE 293	PUSH 14h[sp]	; of a region of RAM, then
012B	CB0006	EE 294	PUSH 06h[sp]	;PUSH a bit pattern to use, then
012E	EF0000	E 295	CALL mem0d	;CALL the Checkerboard Pattern test
		296		; for external memory
		297	BR_ON_ERR Error_Found	
		301	BR_ON_ERR Error_Found	
0136	CB0010	E 302	PUSH 10h[sp]	;PUSH the start and end address
0139	CB0010	EE 303	PUSH 10h[sp]	; of another region of RAM, then
013C	CB0006	EE 304	PUSH 06h[sp]	;PUSH a bit pattern to use, then
013F	EF0000	EE 305	CALL mem0d	;CALL the Checkerboard Pattern test
		306		; for external memory
		307	BR_ON_ERR Error_Found	
		311		
0147	CB0014	E 312	PUSH 14h[sp]	;PUSH a starting address, and
014A	CB0014	EE 313	PUSH 14h[sp]	;PUSH an ending address for
014D	EF0000	EE 314	CALL sys03	; the Program Counter Module
		315		
		316	BR_ON_ERR Error_Found	
		320	BR_ON_ERR Error_Found	
0155	CB0010	E 321	PUSH 10h[sp]	;PUSH the start and end addresses
0158	CB0010	EE 322	PUSH 10h[sp]	; for a second test region for
015B	EF0000	EE 323	CALL sys03	; for the Program Counter Module
		324		
		325	BR_ON_ERR Error_Found	
		329	BR_ON_ERR Error_Found	
0163	C98020		PUSH #2080h	;PUSH the code starting address
0166	CB000A	EE 331	PUSH 0ah[sp]	;PUSH the ending code address
0169	EF0000	EE 332	CALL mem0b	;CALL the Checksum routine
		333		
016C	A1300000	E 334	LD EREG1,#0030h	;ALL DIAG96 TESTS PASSED
		335	BR_ON_ERR Error_Found	; load the appropriate error code
		336		
0170	CF0014	EE 337	CVT POP 14h[sp]	; clean off the stack
0173	65120000	E 338	ADD sp,#12h	
0177	F0	E 339	RET	; return to the calling program
		340		
0178		341	Error_Found:	
		342		
0178	CF0014	E 343	CVT POP 14h[sp]	; clean off the stack
017B	65120000	E 344	ADD sp,#12h	
017F	F0	E 345	RET	; return to the calling program
		346	;*****	
0180		347	end	

```

32 3      END MEM01:
34 3      DECFVBR (DWMJ'DWMJ5) M0RD:
32 1      MEM01: DBOCEDD08E (DWMJ'DWMJ5) DMOED EXLENHVT:

33 3      END MEM02:
33 3      DECFVBR (DWMJ'DWMJ5) M0RD:
33 1      MEM02: DBOCEDD08E (DWMJ'DWMJ5) DMOED EXLENHVT:

```

SYMBOL TABLE LISTING

N A M E	VALUE	ATTRIBUTES
ALL TESTS ASM96	----	MODULE STACKSIZE(20)
ALU01	----	CODE EXTERNAL
ALU02	----	CODE EXTERNAL
ALU03	----	CODE EXTERNAL
ALU04	----	CODE EXTERNAL
ALU05	----	CODE EXTERNAL
BR ON ERR	----	MACRO
D96A96	0000H	CODE REL PUBLIC ENTRY
EREG1	----	REG EXTERNAL
EREG2	----	REG EXTERNAL
ERROR_FOUND	0178H	CODE REL ENTRY
MACRO_TEMP	0000H	REG REL BYTE
MEM01	----	CODE EXTERNAL
MEM02	----	CODE EXTERNAL
MEM03	----	CODE EXTERNAL
MEM04	----	CODE EXTERNAL
MEM05	----	CODE EXTERNAL
MEM06	----	CODE EXTERNAL
MEM07	----	CODE EXTERNAL
MEM08	----	CODE EXTERNAL
MEM09	----	CODE EXTERNAL
MEM0A	----	CODE EXTERNAL
MEM0B	----	CODE EXTERNAL
MEM0C	----	CODE EXTERNAL
MEM0D	----	CODE EXTERNAL
RESET_WATCHDOG	----	MACRO
RI	0006H	NULL ABS
SP	----	REG EXTERNAL
SP_STAT	----	REG EXTERNAL
SPSTATUS	----	MACRO
SPWAIT	----	MACRO
SYS01	----	CODE EXTERNAL
SYS02	----	CODE EXTERNAL
SYS03	----	CODE EXTERNAL
TI	0005H	NULL ABS
ZERO	----	REG EXTERNAL

ASSEMBLY COMPLETED, NO ERROR(S) FOUND.

```

DO:
I
WTTADYD030110000:

```

```

CONFIGURATION IMAGE: B7N00-00-103000000-000 CODE 00000
OPTIONAL MODULES: B7N00-00-103000000-000
SERIES-III B7N-00-01-00000000-00000000-00000000-00000000

```

SERIES-III PL/M-96 V1.0 COMPILATION OF MODULE ALLDIAG96TESTS
 OBJECT MODULE PLACED IN :F2:D96P96.OBJ
 COMPILER INVOKED BY: PLM96.86 :F2:D96P96.P96 CODE DEBUG

```

1      All$Diag96$Tests:
      DO;

      Y22EMBP1 COMPLETED' NO ENCODE(R) LOADED'

      2 1      SYS02: PROCEDURE DWORD EXTERNAL;NVT
      3 2      END SYS02; 0002H NVT V82
      4 1      SYS03: PROCEDURE (parm1,parm2) DWORD EXTERNAL;
      5 2      DECLARE (parm1,parm2) WORD;NVT
      6 2      END SYS03; NVC0
      7 1      ALU01: PROCEDURE DWORD EXTERNAL;NVT
      8 2      END ALU01; 0002H NVT V82
      9 1      ALU02: PROCEDURE DWORD EXTERNAL;
      10 2      END ALU02; 0002H NVT V82
      11 1      ALU03: PROCEDURE DWORD EXTERNAL;NVT
      12 2      END ALU03; 0002H NVT V82
      13 1      ALU04: PROCEDURE (parm1,parm2) DWORD EXTERNAL;
      14 2      DECLARE (parm1,parm2) WORD;NVT
      15 2      END ALU04; 0002H NVT V82
      16 1      ALU05: PROCEDURE (parm1,parm2) DWORD EXTERNAL;
      17 2      DECLARE (parm1,parm2) WORD;NVT
      18 2      END ALU05; 0002H NVT V82
      19 1      MEM01: PROCEDURE DWORD EXTERNAL;NVT
      20 2      END MEM01; 0119H NVT V82
      21 1      MEM02: PROCEDURE DWORD EXTERNAL;NVT
      22 2      END MEM02; 0002H NVT V82
      23 1      MEM03: PROCEDURE DWORD EXTERNAL;NVT
      24 2      END MEM03; 0002H NVT V82
      25 1      MEM04: PROCEDURE (parm1) DWORD EXTERNAL;
      26 2      DECLARE (parm1) WORD;NVT
      27 2      END MEM04; 0002H NVT V82
      28 1      MEM05: PROCEDURE DWORD EXTERNAL;NVT
      29 2      END MEM05; 0002H NVT V82
      30 1      MEM06: PROCEDURE (parm1,parm2) DWORD EXTERNAL;
      31 2      DECLARE (parm1,parm2) WORD;
      32 2      END MEM06;
      33 1      MEM07: PROCEDURE (parm1,parm2) DWORD EXTERNAL;
      34 2      DECLARE (parm1,parm2) WORD;
      35 2      END MEM07;
  
```



```

131 3      EQUH=MEM0D(LMW32ef91f'LMW32ef0b'p1f2b4f01u);
36 1      MEM08: PROCEDURE (parml,param2) DWORD EXTERNAL;
37 2      DECLARE (parml,param2) WORD; p1f2b4f01u);
38 2      END MEM08;
39 3      IF E11012C0Q88'UNMPEL > 322 THEN GOTO end$tests;
39 1      MEM09: PROCEDURE (parml,param2) DWORD EXTERNAL;
40 2      DECLARE (parml,param2) WORD;
41 2      IF END MEM09; UNMPEL > 322 THEN GOTO end$tests;
42 3      EQUH=MEM09(LMW32ef91f'LMW32ef0b);
42 1      MEM0A: PROCEDURE (parml,param2) DWORD EXTERNAL;
43 2      DECLARE (parml,param2) WORD; GOTO end$tests;
44 2      END MEM0A;
45 1      MEM0B: PROCEDURE (parml,param2) DWORD EXTERNAL;
46 2      DECLARE (parml,param2) WORD;
47 2      END MEM0B;
48 3      IF E11012C0Q88'UNMPEL > 322 THEN GOTO end$tests;
48 1      MEM0C: PROCEDURE (parml) DWORD EXTERNAL;
49 2      DECLARE (parml) WORD;
50 2      END MEM0C; UNMPEL > 322 THEN GOTO end$tests;
51 3      EQUH=MEM0C(LMW32ef91f'LMW32ef0b);
51 1      MEM0D: PROCEDURE (parml,param2,param3) DWORD EXTERNAL;
52 2      DECLARE (parml,param2,param3) WORD; GOTO end$tests;
53 2      END MEM0D; UNMPEL > 322 THEN GOTO end$tests;
54 1      DECLARE result DWORD; > 322 THEN GOTO end$tests;
55 1      DECLARE error$codes STRUCTURE (number WORD,detail WORD) AT (.result);
56 3      IF E11012C0Q88'UNMPEL > 322 THEN GOTO end$tests;
56 1      D96P96: PROCEDURE (ram1$start,ram1$stop,
57 2      ram2$start,ram2$stop,
58 3      random$seed,random$length,
59 3      top$of$code,
60 3      argument1,argument2,
61 3      bit$pattern) DWORD PUBLIC; GOTO end$tests;
62 3      EQUH=MEM0D(LMW32ef91f'LMW32ef0b);
62 1      DECLARE (ram1$start,ram1$stop,
63 2      ram2$start,ram2$stop,THEN GOTO end$tests;
64 3      random$seed,random$length,
65 3      top$of$code,
66 3      argument1,argument2, THEN GOTO end$tests;
67 3      EQUH=MEM0D(LMW32ef91f'LMW32ef0b);
67 1      bit$pattern) WORD SLOW;
68 3      IF E11012C0Q88'UNMPEL > 322 THEN GOTO end$tests;
68 2      result=SYS02;
69 2      IF error$codes.number > 255 THEN GOTO end$tests;
70 3      IF E11012C0Q88'UNMPEL > 322 THEN GOTO end$tests;
70 1      result=ALU01;
71 2      IF error$codes.number > 255 THEN GOTO end$tests;
72 2      result=ALU02;
73 2      IF error$codes.number > 255 THEN GOTO end$tests;
74 2      result=ALU03; GOTO end$tests;
75 2      IF error$codes.number > 255 THEN GOTO end$tests;
76 3      IF E11012C0Q88'UNMPEL > 322 THEN GOTO end$tests;
76 1      result=ALU04(47efH,1000H);

```

```

10 3      LEENIT=VTONV(+JCH"TRSHI)
71 2      IF error$codes.number > 255 THEN GOTO end$tests;
73 2      result=ALU05(argument1,argument2);
74 2      IF error$codes.number > 255 THEN GOTO end$tests;
76 2      result=MEM01;
77 2      IF error$codes.number > 255 THEN GOTO end$tests;
79 2      result=MEM02;
80 2      IF error$codes.number > 255 THEN GOTO end$tests;
82 2      result=MEM03;
83 2      IF error$codes.number > 255 THEN GOTO end$tests;

85 2      result=MEM04(0);
86 2      IF error$codes.number > 255 THEN GOTO end$tests;

88 2      result=MEM05;
89 2      IF error$codes.number > 255 THEN GOTO end$tests;
91 2      result=MEM06(ram1$start,ram1$stop);
92 2      IF error$codes.number > 255 THEN GOTO end$tests;

94 2      result=MEM06(ram2$start,ram2$stop);
95 2      IF error$codes.number > 255 THEN GOTO end$tests;

97 2      result=MEM07(ram1$start,ram1$stop);
98 2      IF error$codes.number > 255 THEN GOTO end$tests;

100 2      result=MEM07(ram2$start,ram2$stop);
101 2      IF error$codes.number > 255 THEN GOTO end$tests;

103 2      result=MEM08(ram1$start,ram1$stop);
104 2      IF error$codes.number > 255 THEN GOTO end$tests;

106 2      result=MEM08(ram2$start,ram2$stop);
107 2      IF error$codes.number > 255 THEN GOTO end$tests;

109 2      result=MEM09(ram1$start,ram1$stop);
110 2      IF error$codes.number > 255 THEN GOTO end$tests;

112 2      result=MEM09(ram2$start,ram2$stop);
113 2      IF error$codes.number > 255 THEN GOTO end$tests;

115 2      result=MEM0A(ram1$start,ram1$stop);
116 2      IF error$codes.number > 255 THEN GOTO end$tests;

118 2      result=MEM0A(ram2$start,ram2$stop);
119 2      IF error$codes.number > 255 THEN GOTO end$tests;

121 2      result=MEM0C(bit$pattern);
122 2      IF error$codes.number > 255 THEN GOTO end$tests;

124 2      result=MEM0D(ram1$start,ram1$stop,bit$pattern);
125 2      IF error$codes.number > 255 THEN GOTO end$tests;

127 2      result=MEM0D(ram2$start,ram2$stop,bit$pattern);

```

[illegible]

```

; STATEMENT 56
D96P96:
0000 C800 E PUSH ?FRAME01
0002 A01800 E LD ?FRAME01,SP
; STATEMENT 58
0005 EF0000 E CALL SYS02
0008 A01E02 R LD RESULT+2H,TMP2
000B A01C00 R LD RESULT,TMP0
; STATEMENT 59
000E 89FF0000 R CMP ERRORCODES,#0FFH
0012 D102 BNH @0001
; STATEMENT 60
0014 2226 BR ENDTESTS
; STATEMENT 61
@0001:
0016 EF0000 E CALL ALU01
0019 A01E02 R LD RESULT+2H,TMP2
001C A01C00 R LD RESULT,TMP0
; STATEMENT 62
001F 89FF0000 R CMP ERRORCODES,#0FFH
0023 D102 BNH @0002
; STATEMENT 63
0025 2215 BR ENDTESTS
; STATEMENT 64
@0002:
0027 EF0000 E CALL ALU02
002A A01E02 R LD RESULT+2H,TMP2
002D A01C00 R LD RESULT,TMP0
; STATEMENT 65
0030 89FF0000 R CMP ERRORCODES,#0FFH
0034 D102 BNH @0003
; STATEMENT 66
0036 2204 BR ENDTESTS
; STATEMENT 67
@0003:
0038 EF0000 E CALL ALU03
003B A01E02 R LD RESULT+2H,TMP2
003E A01C00 R LD RESULT,TMP0
; STATEMENT 68
0041 89FF0000 R CMP ERRORCODES,#0FFH
0045 D102 BNH @0004
; STATEMENT 69
0047 21F3 BR ENDTESTS
; STATEMENT 70
@0004:
0049 C9EF47 PUSH #47EFH
004C C90010 PUSH #1000H
004F EF0000 E CALL ALU04
0052 A01E02 R LD RESULT+2H,TMP2
0055 A01C00 R LD RESULT,TMP0
; STATEMENT 71
0058 89FF0000 R CMP ERRORCODES,#0FFH
005C D102 BNH @0005
; STATEMENT 72
005E 21DC BR ENDTESTS
; STATEMENT 73

```



```

0060 CB0008      E      @0005: PUSH ARGUMENT1[?FRAME01]
0063 CB0006      E      PUSH ARGUMENT2[?FRAME01]
0066 EF0000      E      CALL ALU05
0069 A01E02      R      LD RESULT+2H,TMP2
006C A01C00      R      LD RESULT,TMP0
006F 89FF0000    R      ; STATEMENT 74
0073 D102        R      CMP ERRORCODES,#0FFH
0075 21C5        E      BNH @0006
0077 C50015      E      ; STATEMENT 75
0077 EF0000      E      BR ENDTESTS
007A A01E02      R      ; STATEMENT 76
007D A01C00      R      @0006: CALL MEM01
0080 89FF0000    R      LD RESULT+2H,TMP2
0084 D102        R      LD RESULT,TMP0
0086 21B4        E      ; STATEMENT 77
0088 EF0000      E      CMP ERRORCODES,#0FFH
008B A01E02      R      BNH @0007
008E A01C00      R      ; STATEMENT 78
0091 89FF0000    R      BR ENDTESTS
0095 D102        E      ; STATEMENT 79
0097 21A3        E      @0007: CALL MEM02
0099 89FF0000    R      LD RESULT+2H,TMP2
0099 EF0000      R      LD RESULT,TMP0
009C A01E02      R      ; STATEMENT 80
009F A01C00      R      CMP ERRORCODES,#0FFH
00A2 89FF0000    R      BNH @0008
00A6 D102        E      ; STATEMENT 81
00A8 2192        E      BR ENDTESTS
00AA C800        E      ; STATEMENT 82
00AA C800        E      @0008: CALL MEM03
00AC EF0000      R      LD RESULT+2H,TMP2
00AF A01E02      R      LD RESULT,TMP0
00B2 A01C00      R      ; STATEMENT 83
00B5 89FF0000    R      CMP ERRORCODES,#0FFH
00B9 D102        R      BNH @0009
00BB 217F        E      ; STATEMENT 84
00BD EF0000      E      BR ENDTESTS
00BD EF0000      E      ; STATEMENT 85
00C0 A01E02      R      @0009: PUSH R0
00C0 A01E02      R      CALL MEM04
00C3 89FF0000    R      LD RESULT+2H,TMP2
00C3 89FF0000    R      LD RESULT,TMP0
00C3 89FF0000    R      ; STATEMENT 86
00C3 89FF0000    R      CMP ERRORCODES,#0FFH
00C3 89FF0000    R      BNH @000A
00C3 89FF0000    R      ; STATEMENT 87
00C3 89FF0000    R      BR ENDTESTS
00C3 89FF0000    R      ; STATEMENT 88
00C3 89FF0000    R      @000A: CALL MEM05
00C3 89FF0000    R      LD RESULT+2H,TMP2
00C3 89FF0000    R      LD RESULT,TMP0

```

```

00C3 A01C00      R      LD      RESULT,TMP0
                        ;      STATEMENT      89
00C6 89FF0000    R      CMP      ERRORCODES,#0FFH
00CA D102        ;      BNH      @000B
                        ;      STATEMENT      90
00CC 216E        BR      ENDTESTS
                        ;      STATEMENT      91
00CE D185        @000B:  E      PUSH  RAM1START[?FRAME01]
00CE CB0016      E      PUSH  RAM1STOP[?FRAME01]
00D1 CB0014      E      CALL  MEM06
00D4 EF0000      R      LD      RESULT+2H,TMP2
00D7 A01E02      R      LD      RESULT,TMP0
00DA A01C00      ;      STATEMENT      92
                        ;      CMP      ERRORCODES,#0FFH
00DD 89FF0000    R      BNH      @000C
00E1 D102        ;      STATEMENT      93
00E3 2157        BR      ENDTESTS
                        ;      STATEMENT      94
00E5 00E50003    @000C:  E      PUSH  RAM2START[?FRAME01]
00E5 CB0012      E      PUSH  RAM2STOP[?FRAME01]
00E8 CB0010      E      CALL  MEM06
00EB EF0000      R      LD      RESULT+2H,TMP2
00EE A01E02      R      LD      RESULT,TMP0
00F1 A01C00      ;      STATEMENT      95
                        ;      CMP      ERRORCODES,#0FFH
00F4 89FF0000    R      BNH      @000D
00F8 D102        ;      STATEMENT      96
00FA 21400000    E      BR      ENDTESTS
                        ;      STATEMENT      97
00FC 00FC00      @000D:  E      PUSH  RAM1START[?FRAME01]
00FC CB0016      E      PUSH  RAM1STOP[?FRAME01]
00FF CB0014      E      CALL  MEM07
0102 EF0000      R      LD      RESULT+2H,TMP2
0105 A01E02      R      LD      RESULT,TMP0
0108 A01C00      ;      STATEMENT      98
                        ;      CMP      ERRORCODES,#0FFH
010B 89FF0000    R      BNH      @000E
010F D102        ;      STATEMENT      99
                        ;      BR      ENDTESTS
0111 21290000    E      ;      STATEMENT      100
0113 00130000    @000E:  E      PUSH  RAM2START[?FRAME01]
0113 CB0012      E      PUSH  RAM2STOP[?FRAME01]
0116 CB0010      E      CALL  MEM07
0119 EF0000      R      LD      RESULT+2H,TMP2
011C A01E02      R      LD      RESULT,TMP0
011F A01C00      ;      STATEMENT      101
0122 89FF0000    R      CMP      ERRORCODES,#0FFH
0126 D102        BNH      @000F
                        ;      STATEMENT      102
0128 21120000    E      BR      ENDTESTS
                        ;      STATEMENT      103
012A 000A0000    @000F:  E

```

012A	CB0016	E	PUSH	RAM1START[?FRAME01]
012D	CB0014	E	PUSH	RAM1STOP[?FRAME01]
0130	EF0000	E	CALL	MEM08
0133	A01E02	R	LD	RESULT+2H,TMP2
0136	A01C00	R	LD	RESULT,TMP0
			; STATEMENT	104
0139	89FF0000	R	CMP	ERRORCODES,#0FFH
013D	D102		BNH	@0010
			; STATEMENT	105
013F	20FB		BR	ENDTESTS
			; STATEMENT	106
0141	CB0012	E	PUSH	RAM2START[?FRAME01]
0144	CB0010	E	PUSH	RAM2STOP[?FRAME01]
0147	EF0000	E	CALL	MEM08
014A	A01E02	R	LD	RESULT+2H,TMP2
014D	A01C00	R	LD	RESULT,TMP0
			; STATEMENT	107
0150	89FF0000	R	CMP	ERRORCODES,#0FFH
0154	D102		BNH	@0011
			; STATEMENT	108
0156	20E4		BR	ENDTESTS
			; STATEMENT	109
0158	CB0016	E	PUSH	RAM1START[?FRAME01]
015B	CB0014	E	PUSH	RAM1STOP[?FRAME01]
015E	EF0000	E	CALL	MEM09
0161	A01E02	R	LD	RESULT+2H,TMP2
0164	A01C00	R	LD	RESULT,TMP0
			; STATEMENT	110
0167	89FF0000	R	CMP	ERRORCODES,#0FFH
016B	D102		BNH	@0012
			; STATEMENT	111
016D	20CD		BR	ENDTESTS
			; STATEMENT	112
016F	CB0012	E	PUSH	RAM2START[?FRAME01]
0172	CB0010	E	PUSH	RAM2STOP[?FRAME01]
0175	EF0000	E	CALL	MEM09
0178	A01E02	R	LD	RESULT+2H,TMP2
017B	A01C00	R	LD	RESULT,TMP0
			; STATEMENT	113
017E	89FF0000	R	CMP	ERRORCODES,#0FFH
0182	D102		BNH	@0013
			; STATEMENT	114
0184	20B6		BR	ENDTESTS
			; STATEMENT	115
0186	CB0016	E	PUSH	RAM1START[?FRAME01]
0189	CB0014	E	PUSH	RAM1STOP[?FRAME01]
018C	EF0000	E	CALL	MEM0A
018F	A01E02	R	LD	RESULT+2H,TMP2
0192	A01C00	R	LD	RESULT,TMP0
			; STATEMENT	116
0195	89FF0000	R	CMP	ERRORCODES,#0FFH
0199	D102		BNH	@0014

```

019B 209F      ; STATEMENT 117
; BR   ENDTESTS
; STATEMENT 118
@0014:
019D 001C00    E   PUSH  RAM2START[?FRAME01]
019D CB0012    E   PUSH  RAM2STOP[?FRAME01]
01A0 CB0010    E   CALL  MEM0A
01A3 EF0000    R   LD     RESULT+2H,TMP2
01A6 A01E02    R   LD     RESULT,TMP0
01A9 A01C00    R   ; STATEMENT 119
; CMP  ERRORCODES,#0FFH
01AC 89FF0000  R   BNH   @0015
01B0 D102      ; STATEMENT 120
; BR   ENDTESTS
01B2 2088      ; STATEMENT 121
; STATEMENT 121
@0015:
01B4 001C00    E   PUSH  BITPATTERN[?FRAME01]
01B4 CB0004    E   CALL  MEM0C
01B7 EF0000    R   LD     RESULT+2H,TMP2
01BA A01E02    R   LD     RESULT,TMP0
01BD A01C00    R   ; STATEMENT 122
; CMP  ERRORCODES,#0FFH
01C0 89FF0000  R   BNH   @0016
01C4 D102      ; STATEMENT 123
; BR   ENDTESTS
01C6 2074      ; STATEMENT 124
; STATEMENT 124
@0016:
01C8 001C00    E   PUSH  RAM1START[?FRAME01]
01C8 CB0016    E   PUSH  RAM1STOP[?FRAME01]
01CB CB0014    E   PUSH  BITPATTERN[?FRAME01]
01CE CB0004    E   CALL  MEM0D
01D1 EF0000    R   LD     RESULT+2H,TMP2
01D4 A01E02    R   LD     RESULT,TMP0
01D7 A01C00    R   ; STATEMENT 125
; CMP  ERRORCODES,#0FFH
01DA 89FF0000  R   BNH   @0017
01DE D102      ; STATEMENT 126
; BR   ENDTESTS
01E0 205A      ; STATEMENT 127
; STATEMENT 127
@0017:
01E2 001C00    E   PUSH  RAM2START[?FRAME01]
01E2 CB0012    E   PUSH  RAM2STOP[?FRAME01]
01E5 CB0010    E   PUSH  BITPATTERN[?FRAME01]
01E8 CB0004    E   CALL  MEM0D
01EB EF0000    R   LD     RESULT+2H,TMP2
01EE A01E02    R   LD     RESULT,TMP0
01F1 A01C00    R   ; STATEMENT 128
; CMP  ERRORCODES,#0FFH
01F4 89FF0000  R   BNH   @0018
01F8 D102      ; STATEMENT 129
; BR   ENDTESTS
01FA 2040      ; STATEMENT 130
; STATEMENT 130
@0018:
01FC 001C00    E   PUSH  RAM1START[?FRAME01]
01FC CB0016    E   PUSH  RAM1STOP[?FRAME01]
01FF CB0014    E   CALL  SYS03
0202 EF0000    E

```



```

0205 A01E02      R      LD      RESULT+2H,TMP2
0208 A01C00      R      LD      RESULT,TMP0
                        ; STATEMENT 131
020B 89FF0000    R      CMP     ERRORCODES,#0FFH
020F D102        31     BNH     @0019
                        ; STATEMENT 132
0211 2029        32     BR      ENDTESTS
                        ; STATEMENT 133
0213 @0019:      33     @0019:
0213 CB0012      33     E      PUSH  RAM2START[?FRAME01]
0216 CB0010      33     E      PUSH  RAM2STOP[?FRAME01]
0219 EF0000      34     E      CALL  SYS03
021C A01E02      35     R      LD      RESULT+2H,TMP2
021F A01C00      36     R      LD      RESULT,TMP0
                        ; STATEMENT 134
0222 89FF0000    36     R      CMP     ERRORCODES,#0FFH
0226 D102        37     BNH     @001A
                        ; STATEMENT 135
0228 2012        38     BR      ENDTESTS
                        ; STATEMENT 136
022A @001A:      39     @001A:
022A C98020      39     E      PUSH  #2080H
022D CB000A      39     E      PUSH  TOPOFCODE[?FRAME01]
0230 EF0000      39     E      CALL  MEM0B
0233 A01E02      39     R      LD      RESULT+2H,TMP2
0236 A01C00      39     R      LD      RESULT,TMP0
                        ; STATEMENT 137
0239 ADF000      39     R      LDBZ   ERRORCODES,#0F0H
                        ; STATEMENT 138
023C @001A:      39     R      LD      TMP2,RESULT+2H
023C A0021E      39     R      LD      TMP0,RESULT
023F A0001C      39     E      POP   ?FRAME01
0242 CC00        39     E      LD      TMP6,[SP]
0244 A21822      39     E      ADD   SP,#16H
0247 65160018    39     E      BR     [TMP6]
024B E322        39     ; STATEMENT 139
                        ; STATEMENT 140
024B E322        39     END

```

```

MODULE INFORMATION:
CODE AREA SIZE      = 024DH   589D
CONSTANT AREA SIZE  = 0000H   0D
DATA AREA SIZE      = 0000H   0D
STATIC REGS AREA SIZE = 0004H   4D
OVERLAYABLE REGS AREA SIZE = 0000H   0D
MAXIMUM STACK SIZE  = 000AH   10D
183 LINES READ

```

PL/M-96 COMPILATION COMPLETE. 0 WARNINGS, 0 ERRORS

BT\N-96 COMBINATION COMPLETE 0 AVERAGING 8 ENDS

193 CYCLE KWD
MCS-96 MACRO ASSEMBLER SELECTED TESTS_ASM96.D
OASHTVAVRTE BUCS VKEY EISE = 8888H 8D
SERIES-III MCS-96 MACRO ASSEMBLER, V1.0 4D
DAILY VKEY EISE = 8888H 8D
SOURCE FILE: F5:D96FST.A96 = 8888H 8D
OBJECT FILE: F5:D96FST.OBJ = 854DH 200D
CONTROLS SPECIFIED IN INVOCATION COMMAND: NOGEN DEBUG
MODULE INFORMATION:

ERR LOC	OBJECT	LINE	SOURCE STATEMENT
		1	
		2	;*****
		3	Selected Tests_ASM96 MODULE STACKSIZE(20)
		4	;***** 0031
		5	; ; ;
0548	0533	6	; In order to run this module, the STACK must be ALL external. and the
0541	02100019	7	; data ram partitioned for memory test must not include ANY of the STACK
0544	V51033	8	; ; ;
0543	CC88	9	; To call this module
0536	V0001C	10	; ; ;
053C	V0031E	11	; ; ;
		12	; ; ;
0530	V00000	13	; ; ;
		14	; ; ;
		15	; ; ;
0530	V01C00	16	; ; ;
0533	V01E03	17	; ; ;
0530	E00000	18	; ; ;
053D	C0000V	19	; ; ;
053V	C00030	20	; ; ;
053V		21	; ; ;
		22	; ; ;
0530	5035	23	; ; ;
		24	;*****
0530	D105	25	; ; ;
0000	0535	26	; ; ;
		27	; ; ;
0536	V01C00	28	; ; ;
053C	V01E03	29	; ; ;
0530	E00000	30	; ; ;
0000	0530	31	; ; ;
0533	C00015	32	; ; ;
0533		33	; ; ;
		34	; ; ;
0533	5030	35	; ; ;
		36	; ; ;
0506	D105	37	; ; ;
0508	000000		; ; ;
			; ; ;
0500	V01C00		; ; ;
0502	V01E03		; ; ;

ERR	LOC	OBJECT	E	LINE	SOURCE STATEMENT
	0000	C00074	E	38	
				39	\$include (:f3:dstmac.inc)
				40	*****
			=1	41	;DST_Macros INCLUDE FILE ;*****
	001V	E00000	E	42	*****
	001A	C00078	E	43	
	0000	C00078	E	44	rseq
			=1	45	
	0000		=1	46	macro temp: DSB 1
			=1	47	extrn zero,sp_stat
	002C	E00000	E	48	
	0000	C00074	E	49	cseg
	0005	C00074	E	50	ti equ 5
	0006		=1	51	ri equ 6
			=1	52	
			=1	53	
	002C	E00000	E	54	*****
			=1	55	RESET_WATCHDOG MACRO ;macro to reset the watchdog
			=1	56	LDB 0ah,#leh
			=1	57	LDB 0ah,#0elh
	0023	E00000	E	58	ENDM
			=1	59	*****
			=1	60	
			=1	61	
	0040	E00000	E	62	*****
			=1	63	SPSTATUS MACRO v1 ; macro to read sp_stat to
			=1	64	LOCAL Sp_read ; work around the ri/ti bug
			=1	65	Sp_read: ; on the 8x9x-90.
	003E	E00000	E	66	LDB macro temp,sp_stat
	003B	C00000	E	67	ORB v1,macro temp
	003B	C00000	E	68	ANDB macro temp,#01100000B
			=1	69	JNE Sp_read
			=1	70	ENDM
			=1	71	*****
	005E	E00000	E	72	
	005B	C00000	E	73	
	005B	C00000	E	74	
			=1	75	*****
			=1	76	SPWAIT MACRO v2 ; macro to wait for ri/ti set
			=1	77	; and avoid 8x9x-90 bug.
	007E	E00000	E	78	; NOTE!! this macro won't work
			=1	79	; with a full duplex line.
			=1	80	JBC sp_stat,v2,\$
			=1	81	LDB zero,sp_stat
	0074	E00000	E	82	ENDM
			=1	83	*****
			=1	84	
			=1	85	
	006V	E00000	E	86	*****
			=1	87	*****
			=1	88	BR_ON_ERR MACRO Label ; macro to test high byte of
			=1	89	; EREG1 and branch away if
	0080	E00000	E	90	CMPB ereql+lh,zero ; the byte is non-zero (which
			=1	91	BNE Label ; means there was an error).
			=1	92	ENDM
			=1	93	*****
	0080	OB10CL	=1	94	*****

ERR LOC	OBJECT	LINE	SOURCE STATEMENT	
		96		
	0000	97	D96FST:	
		98		
	0000 EF0000	E 99	CALL sys02.A96	;CALL the Power Up Test
		100		
		101	BR_ON_ERR	Error_Found
		105		
	000A EF0000	E 106	CALL alu01	;CALL the Add/Subtract test
		107		
		108	BR_ON_ERR	Error_Found
		112		
	0014 EF0000	E 113	CALL alu02	;CALL the MULUB test
		114		
		115	BR_ON_ERR	Error_Found
		119		
	001E EF0000	E 120	CALL alu03	;CALL the Multiply/Divide Table
		121		; driven test
		122	BR_ON_ERR	Error_Found
		126		
	0028 CB000C	E 127	PUSH 06h[sp]	;PUSH a seed and test length
	002B CB000C	E 128	PUSH 06h[sp]	; for the random number based
	002E EF0000	E 129	CALL alu04	; Multiply/Divide Random test
		130		
		131	BR_ON_ERR	Error_Found
		135		
	0038 CB0006	E 136	PUSH 06h[sp]	;PUSH an argument
	003B CB0006	E 137	PUSH 06h[sp]	;PUSH another argument
	003E EF0000	E 138	CALL alu05	;CALL the Multiply/Divide Core test
		139		
		140	BR_ON_ERR	Error_Found
		144		
	0048 EF0000	E 145	CALL mem01	;CALL the Complementary Address test
		146		; for internal registers
		147	BR_ON_ERR	Error_Found
		151		
	0052 EF0000	E 152	CALL mem03	;CALL the Galloping 1s/0s test
		153		; for internal registers
		154	BR_ON_ERR	Error_Found
		158		
	005C EF0000	E 159	CALL mem05	;CALL the Checkerboard Pattern test
		160		; for internal registers
		161	BR_ON_ERR	Error_Found
		165		
	0066 CB0014	E 166	PUSH 14h[sp]	;PUSH a start and end address
	0069 CB0014	E 167	PUSH 14h[sp]	; for a region of RAM to conduct
	006C EF0000	E 168	CALL mem06	; the Complementary Address test for
		169		; external RAM
		170	BR_ON_ERR	Error_Found
		174		
	0074 CB0010	E 175	PUSH 10h[sp]	;PUSH a start and end address
	0077 CB0010	E 176	PUSH 10h[sp]	; for another region of RAM to conduct
	007A EF0000	E 177	CALL mem06	; the Complementary Address test for
		178		; external RAM
		179	BR_ON_ERR	Error_Found
		183		
	0082 CB0014	E 184	PUSH 14h[sp]	;PUSH a start and end address
	0085 CB0014	E 185	PUSH 14h[sp]	; for a region of RAM, and PUSH

ERR LOC	OBJECT	LINE	SOURCE STATEMENT	
0088	CB0006	E 186	PUSH 06h[sp]	; a bit pattern to use in the
008B	EF0000	E 187	CALL mem0d	; Checkerboard Pattern test for
				; external RAM
189 BR_ON_ERR Error_Found				
193				
0093	CB0010	E 194	PUSH 10h[sp]	; PUSH a start and end address for
0096	CB0010	E 195	PUSH 10h[sp]	; another region of RAM, and PUSH
0099	CB0006	E 196	PUSH 06h[sp]	; a bit pattern to use in the
009C	EF0000	E 197	CALL mem0d	; Checkerboard Pattern test for
198				; external RAM
199 BR_ON_ERR Error_Found				
203				
00A4	CB0014	E 204	PUSH 14h[sp]	; PUSH a start and end address
00A7	CB0014	E 205	PUSH 14h[sp]	; for a region of RAM to conduct
00AA	EF0000	E 206	CALL sys03	; the Program Counter test
207				
208 BR_ON_ERR Error_Found				
212				
00B2	CB0010	E 213	PUSH 10h[sp]	; PUSH a start and end address
00B5	CB0010	E 214	PUSH 10h[sp]	; for another region of RAM to conduct
00B8	EF0000	E 215	CALL sys03	; the Program Counter test
216				
217 BR_ON_ERR Error_Found				
221				
00C0	C98020	E 222	PUSH #2080h	; PUSH the code starting address
00C3	CB000A	E 223	PUSH 0ah[sp]	; PUSH the code ending address
00C6	EF0000	E 224	CALL mem0b	; CALL the checksum routine
225				
00C9	A1310000	E 226	CODE LD eregl, #0031h	; ALL DIAG96 TESTS PASSED
227				; load appropriate error code
228				
00CD	CF0014	E 229	POP 14h[sp]	; clean off the stack
00D0	65120000	E 230	ADD sp, #12h	
00D4	F0	E 231	RET	; return to the calling program
232				
00D5		E 233	Error_Found:	
234				
00D5	CF0014	E 235	POP 14h[sp]	; clean off the stack
00D8	65120000	E 236	ADD sp, #12h	
00DC	F0	E 237	RET	; return to the calling program
238				
239 *****				
00DD		E 240	end	

SYMBOL TABLE LISTING

N A M E	VALUE	ATTRIBUTES	
ALU01	37	CODE EXTERNAL	
ALU02	37	CODE EXTERNAL	
ALU03	37	CODE EXTERNAL	
ALU04	37	CODE EXTERNAL	
ALU05	37	CODE EXTERNAL	
BR_ON_ERR	37	MACRO	
D96FST	370000H	CODE REL PUBLIC ENTRY	
EREG1	37	REG EXTERNAL	
EREG2	37	REG EXTERNAL	
ERROR_FOUND	3700D5H	CODE REL ENTRY	
MACRO_TEMP	370000H	REG REL BYTE	
MEM01	37	CODE EXTERNAL	
MEM02	37	CODE EXTERNAL	
MEM03	37	CODE EXTERNAL	
MEM04	37	CODE EXTERNAL	
MEM05	37	CODE EXTERNAL	
MEM06	37	CODE EXTERNAL	
MEM07	37	CODE EXTERNAL	
MEM08	37	CODE EXTERNAL	
MEM09	37	CODE EXTERNAL	
MEM0A	37	CODE EXTERNAL	
MEM0B	37	CODE EXTERNAL	
MEM0C	37	CODE EXTERNAL	
MEM0D	37	CODE EXTERNAL	
RESET_WATCHDOG	37	MACRO	
RI	370006H	NULL ABS	
SELECTED_TESTS_ASM96	37	MODULE STACKSIZE(20)	
SP	37	REG EXTERNAL	
SP_STAT	37	REG EXTERNAL	
SPSTATUS	37	MACRO	
SPWAIT	37	MACRO	
SYS01	37	CODE EXTERNAL	
SYS02	37	CODE EXTERNAL	
SYS03	37	CODE EXTERNAL	
TI	370005H	NULL ABS	
ZERO	37	REG EXTERNAL	
ASSEMBLY COMPLETED, NO ERROR(S) FOUND.			
8888 888888	8	888	888
8888 888888	8	888	888
888 888 888888	8	888	888

MCS-96 MACRO ASSEMBLER DSTUSR

SERIES-III MCS-96 MACRO ASSEMBLER, V1.0

SOURCE FILE: F2:DSTUSR.A96

OBJECT FILE: F2:DSTUSR.OBJ

CONTROLS SPECIFIED IN INVOCATION COMMAND: GEN DEBUG

ERR	LOC	OBJECT	LINE	SOURCE STATEMENT
			1	*****
			2	DSTUSR MODULE main,stacksize(2)
			3	*****
			4	
			5	
			6	oseg at 40h
			7	User_Registers: DSB lch
			8	temp set User_Registers:WORD
			9	
			10	rseg
			11	EXTRN sp,zero,timer1,ereql
			12	
			13	
			14	dseg at 100h ;to ensure that the STACK does not get
			15	
			16	DSEG1: DSB 700h ; located in an area of RAM that will be
			17	
			18	dseg at 4200h ; memory tested, reserve those regions
			19	
			20	DSEG2: DSB 1e00h ; as data segments.
			21	
			22	
			23	
			24	cseg at 2080h
			25	
			26	extrn alu04,alu01,alu02,mem06,mem0a,error_proc,alu05
			27	EXTRN DSTISR
			28	
			29	LD temp,#0ffh
			30	DJNZ temp,\$; wait for sbe96 NMIs to stop
			31	
			32	LD sp,#STACK
			33	
			34	PUSH #100h ;RAM segment1 start address
			35	PUSH #7ffh ;RAM segment1 end address
			36	PUSH #4200h ;RAM segment2 start address
			37	PUSH #5fffh ;RAM segment2 end address
			38	PUSH #47efh ;random seed
			39	PUSH #1000h ;random test length
			40	PUSH #3fffh ;top of code address
			41	PUSH #9d42h ;an argument for mul/div test
			42	PUSH #778ch ;another argument for mul/div test
			43	PUSH #5a5ah ;bit pattern for memory test
			44	CALL DSTISR ;CALL the Dynamic Stability Test
			45	
			46	
			47	Main_Task:
			48	
			49	push #8080h ; use the multiply/divide core
			50	push #8000h ; test on the arguments
			51	call alu05

ERR	LOC	OBJECT	LINE	SOURCE STATEMENT	
	20B5	980001	E 52	cmpb ereql+1,zero	; #8080h and #8000h in all
	20B8	DF022074	53	bne error_found	; combinations
				1	
				IJE \$+4	
				ISJMP error_found	; combinations
	20BC	C90080	54		
	20BF	C90080	55	push #8000h	
	20C2	EF0000	56	push #8080h	
	20C5	980001	E 57	call alu05	
	20C8	DF022064	E 58	cmpb ereql+1,zero	
			59	bne error_found	
				1	
				IJE \$+4	
				ISJMP error_found	
	20CC	C90080	60		
	20CF	C90080	61	push #8000h	
	20D2	EF0000	62	push #8000h	
	20D5	980001	E 63	call alu05	
	20D8	D756	E 64	cmpb ereql+1,zero	
			65	bne error_found	
	20DA	C98080	66		
	20DD	C90080	67	push #8080h	
	20E0	EF0000	68	push #8000h	
	20E3	980001	E 69	call alu05	
	20E6	D748	E 70	cmpb ereql+1,zero	
			71	bne error_found	
	20E8	C90001	72		
	20EB	C9FF07	73	push #100h	; perform a galloping 1s/0s test
	20EE	EF0000	74	push #7ffh	; on a small section of RAM
	20F1	980001	E 75	Call mem0a	
	20F4	D73A	E 76	cmpb ereql+1,zero	
			77	bne error_found	
	20F6	C800	78		
	20F8	C90020	E 79	push timer1	; send a timer1 based seed to the
	20FB	EF0000	80	push #2000h	; random number based multiply/divide
	20FE	980001	E 81	call alu04	; test and let it run for a string
	2101	D72D	E 82	cmpb ereql+1h,zero	; of 2000h argument pairs
			83	bne error_found	
	2103	EF0000	84		
	2106	980001	E 85	call alu01	; perform the add/subtract test
	2109	D725	E 86	cmpb ereql+1h,zero	
			87	bne error_found	
	210B	C90042	88		
	210E	C9FF5F	89	push #4200h	; perform a Complementary address test
	2111	EF0000	90	push #5fffh	; on a large section of RAM
	2114	980001	E 91	Call mem06	
	2117	D717	E 92	cmpb ereql+1,zero	
			93	bne error_found	
	2119	EF0000	94		
	211C	980001	E 95	call alu02	; perform the MULUR test
	211F	D70F	E 96	cmpb ereql+1h,zero	
			97	bne error_found	
	2121	C800	98		
	2123	C90020	E 99	push timer1	; send another timer1 based seed to
	2126	EF0000	100	push #2000h	; the random number based multiply/
	2129	980001	E 101	call alu04	; divide test
	212C	D702	E 102	cmpb ereql+1h,zero	
			103	bne error_found	
			104		

MCS-96 MACRO ASSEMBLER DSTUSR

ERR	LOC	OBJECT	LINE	SOURCE STATEMENT	
	212E	277C	105	BR Main_task	; start the main_task tests over
			106		
	2130		107	Error_found:	
	2130	FA	108	di	; if an error is found, disable
	2131	EF0000	109	CALL Error_Proc	; interrupts and call the error
			110		; procedure in the DST96.LIB.
			111		; the test that found an error will
			112		; have placed the appropriate
			113		; error codes in locations EREG1 and
			114		; EREG2 for output through Error_Proc
	2134	27FE	115	BR \$	
			116	;*****	
			117		
	2136		118	end	

M S-96 MACRO ASSEMBLER DSTUSR

SYMBOL TABLE LISTING

N A M E	VALUE	ATTRIBUTES
ALU01	----	CODE EXTERNAL
ALU02	----	CODE EXTERNAL
ALU04	----	CODE EXTERNAL
ALU05	----	CODE EXTERNAL
DSEG1	0100H	DATA ABS BYTE
DSEG2	4200H	DATA ABS BYTE
DSTISR.	----	CODE EXTERNAL
DSTUSR.	----	MODULE MAIN STACKSIZE(2)
EREG1	----	REG EXTERNAL
ERROR FOUND	2130H	CODE ABS ENTRY
ERROR PROC.	----	CODE EXTERNAL
MAIN TASK	20ACH	CODE ABS ENTRY
MEM06	----	CODE EXTERNAL
MEM0A	----	CODE EXTERNAL
SP.	----	REG EXTERNAL
TEMP.	0040H	OVERLAY ABS WORD
TIMER1.	----	REG EXTERNAL
USER REGISTERS.	0040H	OVERLAY ABS BYTE
ZERO.	----	REG EXTERNAL

ASSEMBLY COMPLETED, NO ERROR(S) FOUND.

80960 Article Reprints

8



Intel's 80960: An Architecture Optimized for Embedded Control

The 80960 architecture is designed to support a wide range of embedded control applications. It offers a high level of performance, flexibility, and ease of use. The architecture is optimized for embedded control applications, allowing system designers to take advantage of silicon technology advancements without architectural limits.

- an optimized instruction set
- a flexible interrupt structure
- a user-supervisor model
- a powerful fault-handling structure, and
- multiprocessing hooks and debug support.

Intel's internally developed 80960 architecture allows embedded system designers to take advantage of silicon technology advancements without architectural limits. The 80960, developed from scratch for embedded control applications, eliminates architectural obstacles to state-of-the-art implementation techniques that allow parallel and out-of-order instruction execution.

In introducing the 80960 architecture, I distinguish between the architecture and the microarchitecture of an implementation. A microarchitecture is an actual implementation of the architecture's instruction set and programming resources. Different microarchitectures may have different pipeline construction, internal bus widths, register set porting, cache parameterization (two-way, four-way, and so on), and degrees of parallelism, or may not execute instructions out of order. The architecture is specified in such a way that wide latitude is available for future microarchitectural advancements. In this way both very high performance and highly integrated controllers can be built using the 80960 architecture.

Principally, the 80960 architecture allows, for at least the next decade, silicon technology and microarchitectural advances to translate directly into increased performance without architectural limitations. While the common performance target of typical RISC architectures is an execution speed of one instruction per processor clock cycle, the 80960 architecture targets the execution of multiple instructions per clock cycle (fractional clock cycles per instruction). By defining an architecture that supports parallel instruction execution and out-of-order instruction execution, we do not constrain performance advances to the system clock cycle.

Additionally, the 80960 has been optimized for the wide range of applications that are unencumbered by a need to be compatible with an existing embedded control architecture. These applications are often very cost sensitive, require a different mix of instructions than reprogrammable applications, have demanding interrupt response requirements, and use real-time executives rather than full-blown operating systems. With these factors in mind, we developed the 80960 while avoiding architectural constructs that would restrict an implementation's capability to execute multiple instructions in one clock cycle.

Executing instructions in one clock cycle is not fast enough for this standard-core architecture. Its parallelism and out-of-order execution promise fractional clock rates in future implementations.

David P. Ryan
Intel Corporation

80960 architecture

The architecture also allows application-specific extensions such as:

- instruction set extensions (floating-point operations),
- special registers,
- larger caches,
- multiple caches,
- on-chip program and data memory,
- a memory management and protection unit,
- fault-tolerance support,
- multiprocessing support, and
- real-time peripherals (DMA, analog/digital, serial ports).

The 80960's instruction set has also been optimized for embedded control applications. It offers Boolean operations more powerful than those of the 8051 family. Single instructions access frequently executed functions for increased code density and performance. They include CALL, RET, Compare_and_Branch, Conditional_Compare, Compare_and_Increment or Decrement, and Bit Field Extract.

A priority interrupt structure simplifies the management of real-time events, and, along with a user/supervisor model, supports fast, safe, real-time kernel operation. A generalized fault-handling mechanism simplifies the task of detecting errant arithmetic calculations or other conditions that typically require a significant amount of user code runtime support. The 80960 does not require sophisticated, optimizing high-

level-language compilers to achieve high performance. However, no obstacles to performance enhancements via a good optimizing compiler exist.

Since products based on the 80960 have high performance levels, even without code optimization, many users will attain their required system performance with 80960 products without having to understand the parallelism of the implementation they are using.

Architecture overview

The 80960 can best be described as a register-rich, load/store architecture with an instruction set designed to let implementations exploit pipelining and parallel execution strategies. Direct embedded control support includes:

- an optimized instruction set,
- a flexible interrupt structure,
- a user-supervisor model,
- a powerful fault-handling structure, and
- multiprocessing hooks and debug support.

The architecture extends easily.

Figure 1 shows a logical block diagram of the architecture's programming resources. The 32-bit memory space is flat. Data moves between memory and registers via load and store instructions. Processing elements surrounding the registers manipulate parallel data; they receive their instructions from the

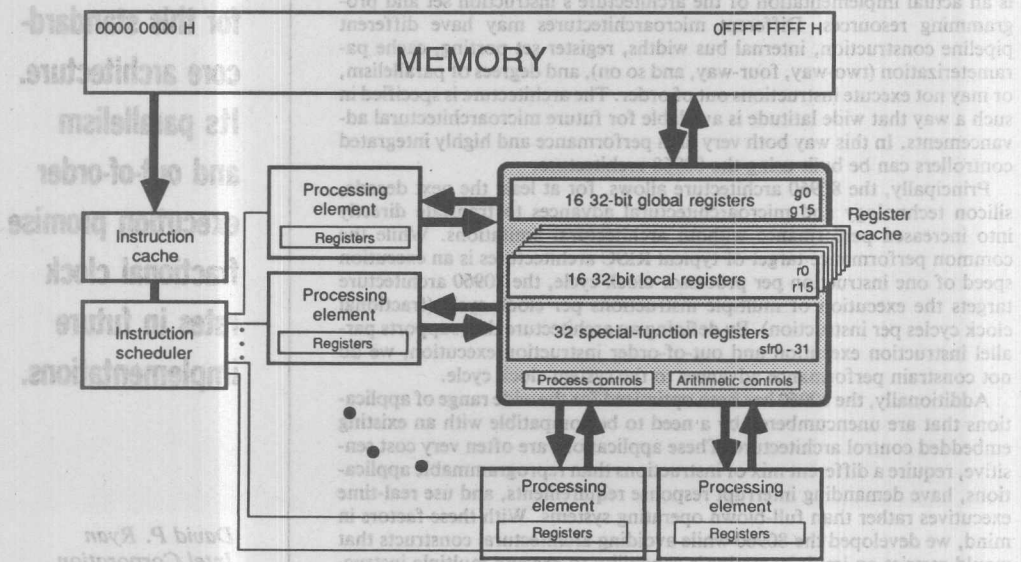


Figure 1. Block diagram of the 80960 architecture.

instruction sequencer.

The instruction sequencer reads multiple instructions simultaneously from an instruction cache and presents the instructions in parallel to the processing elements as appropriate. When the instruction stream or an asynchronous event requires a context switch, the local variables from the suspended procedure move to the register cache. Memory accesses to save previously suspended register sets occur only when the register cache is filled. The implementation determines the number of architecturally transparent register sets that can be cached.

We expect different implementations of the processing elements in an 80960-based controller—optimized for specific applications—will evolve. We defined a standard core architecture to maintain binary-compatible instruction sets across all implementations for leveraging compiler and real-time kernel development. Subsequent references to the 80960, without an alpha suffix, refer to the architecture. References to an 80960.XY pertain to actual implementations of the core architecture, which may contain architectural extensions and/or on-chip peripherals. (See the accompanying box for a discussion of three implementations.)

Implementations of the 80960

As an example of an actual implementation of the core architecture, consider the first 80960 implementation, the 80960KB controller. The KB provides architectural extensions such as floating-point operations (Figure A). Its on-chip floating-point unit implements the IEEE floating-point standard, including transcendental support. Floating-point performance exceeds 4 million Whetstone operations per second at 20 MHz. The 80960KB integrates a 512-byte instruction cache and an interrupt controller on chip.

Another member of the family, the 80960KA controller, fits the KB socket but lacks the on-chip floating-point unit. The 80960MC controller, a military-qualified version similar to the KB, adds Ada tasking support and a memory management and protection unit.

The 80960KA, KB, and MC microarchitecture sustains execution of up to one instruction per clock cycle at 20 MHz (20 native MIPS). It performs a variety of benchmark programs seven to 10 times faster than a VAX 11/780 (7 to 10 VAX MIPS).

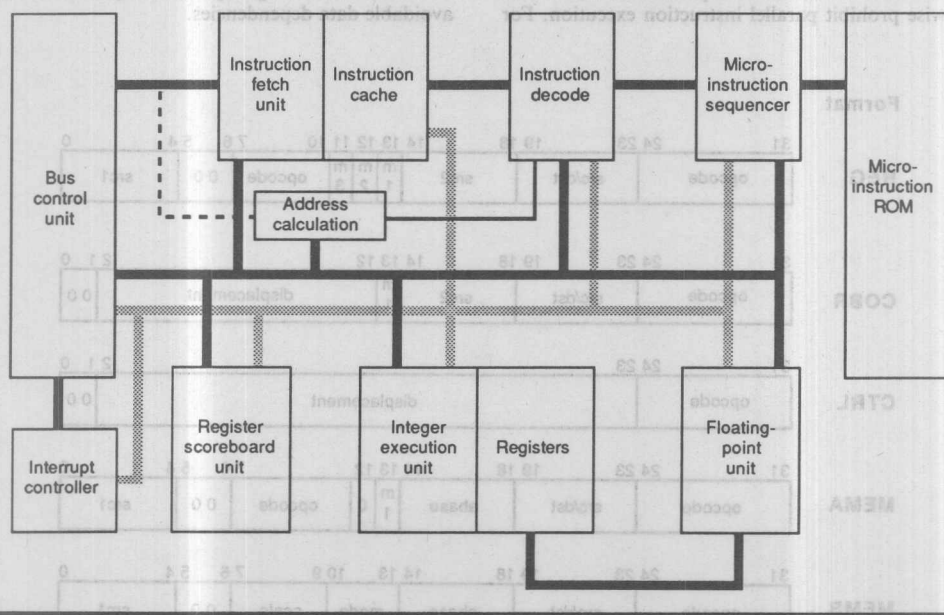


Figure A. Block diagram of the 80960KB controller.

Core instruction set

The 80960 instruction set is similar in design to engines in reduced instruction set computers. Because the 80960 was designed to avoid performance bottlenecks resulting from instruction decoding times, all opcodes are 32 bits (one word) in length and must be aligned on word boundaries. The only two-word (64-bit) instruction format loads 32-bit immediate constants and assists effective address calculations. Generally, load, store, and control instructions access memory. All other instructions access only registers.

Thirty-two-bit opcodes provide tremendous flexibility in instruction encoding. However, performance penalties associated with code size can occur when often-used complex instruction sequences are not available in a one-word format. Large code size not only increases system cost due to larger memory requirements but also results in penalties in cache utilization and bus-bandwidth requirements.

The 80960 includes one-word multifunction instructions to avoid such code density problems that increase memory requirements and to allow complex functions to be parallelized. For example, the CALL and RET instructions provide one-word encodings for sequences that otherwise require several instructions. Implementations of the architecture could perform the CALL/RET operations in parallel with other instruction execution. Or, the processor could execute from an on-chip ROM a similar sequence of simple 80960 instructions that the user would have to write if CALL/RET instructions were not in the architecture. Executing the code from permanent on-chip storage results in higher performance than does requiring the instructions to be fetched and cached every time they are used. In addition to ensuring higher performance, or possibly parallel performance, implementations of such functions as ROMed assembly code sequences—triggered by a one-word opcode—are more silicon efficient than are increases of on-chip cache sizes to deal with poor code density. For example, a ROM cell is typically one fourth the size of a RAM cell.

The availability of complex instructions in the architecture does not prohibit the programmer from bypassing them if simpler functions are desired. For example, a BAL (Branch and Link) instruction can be used to call a procedure that does not require a new set of local registers. The BAL instruction saves the next instruction pointer in a register and branches to the specified destination. When the procedure is ready to return, it executes an indirect branch to the return instruction pointer that was saved. It is likely that BAL will always be faster than CALL since no local registers are being saved. The programmer can use whichever method best suits the circumstances.

Figure 2 shows the 80960 instruction encodings. Most instructions appear in the REG format, which specifies an opcode and three registers/literals (one of 32 registers, or a constant value in the range 0 to 31).

The COBR format specifies a set of compare and branch instructions. The CRTL format covers branch and call instructions. The MEM formats support load and store instructions.

Much of the instruction set is what one would expect to encounter in all processors (ADD, MUL, SHIFT, BRANCH); however, some instructions deserve special mention.

- The register-register move instructions transfer one, two, three, or four register values. The same is true for the load and store instructions (for example, LDQ loads four words into four registers).
- In addition to the normal shift instructions, the SHRDI instruction provides an adjustment to the result so the instruction can be used to divide a value by a power of two. (Normal right-shift instructions do not divide correctly when the value is negative and odd.)
- All logical operations of two operands are provided (AND, NOTAND, ANDNOT, NOR).
- An extensive set of bit instructions exists (SET BIT, CLEAR BIT, NOT BIT, SCAN a register for the first 0, or 1, BRANCH on bits set or clear), as well as instructions for accessing bit fields (MODIFY, EXTRACT).
- Single-instruction COMPARE AND BRANCH encodings optimize code density for these frequently executed operations.
- Conditional compare (CONCOMP) instructions speed bounds checking.

Table 1 on the next page summarizes the 80960 core architecture instruction set.

The architecture directly supports integer (signed) and ordinal (unsigned) data types. Bits, bit fields, bytes, short words, words, and double words can be manipulated in registers and transferred to and from memory. Triple words and quad words can also move between the registers and memory.

Register operations

Most 80960 instructions operate on registers. The architecture provides arithmetic, logical, bit and fit field, data movement, and comparison operations. To take full advantage of the large register set, three-operand instructions specify any register as one or both sources and/or the destination of an operation.

Arithmetic and logical. The architecture supports both standard and extended arithmetic operations. Add, subtract, multiply, and divide operations are available on 32-bit integers and ordinals. Extended multiply operates on two 32-bit ordinals and generates a 64-bit result. Extended divide divides a 64-bit ordinal by a 32-bit ordinal, producing a 32-bit quotient and 32-bit remainder. Addition and subtraction with carry allow 32-bit ordinals to provide extended precision adds and subtracts.

80960 architecture

Table 1.
80960 instruction summary.

REGISTER OPERATIONS		CONTROL OPERATIONS	
ARITHMETICS		BRANCH	
add[i o]	Add	b	Branch (± 2 MByte relative offset)
addc	Add with Carry	bx	Branch Extended (32-Bit Indirect Branch)
sub[i o]	Subtract	bal[x]	Branch and Link ("RISC Branch")
subc	Subtract with Borrow	bt[*]	Branch on Condition
mull[i o]	Multiply	cmpib[*]	Compare Integer and Branch on Condition
emul	Extended Multiply	cmpob[*]	Compare Ordinal and Branch on Condition
div[i o]	Divide	FAULT	
ediv	Extended Divide	fault[*]	Fault on Condition
rem[i o]	Remainder	syncf	Synchronize Faults
modi	Modulo Integer	PROCEDURE	
sh[lo ro li ri di]	Shift	call	Procedure Call (± 2 MByte relative offset)
MOVEMENT		callx	Call Extended (32-Bit Indirect Call)
mov[i t q]	Move registers to registers	calls	System Procedure Call
lda	Load Address	ret	Return
COMPARISON		ENVIRONMENT	
cmp[i o]	Compare	modpc	Modify Process Controls
cmpdec[i o]	Compare and Decrement	modac	Modify Arithmetic Controls
cmpinc[i o]	Compare and Increment	modtc	Modify Trace Controls
concmp[i o]	Conditional Compare	flushregs	Flush Local Register Cache to Memory
test[*]	Test for condition	DEBUG	
scanbyte	Scan for matching byte	mark	Conditionally generate Trace Fault
LOGICAL		fmark	Unconditionally generate Trace Fault
and	$\text{dst} := \text{src1} \& \text{src2}$	MEMORY OPERATIONS	
andnot	$\text{dst} := \text{src2} \& (\sim \text{src1})$	LOAD/STORE	
notand	$\text{dst} := (\sim \text{src2}) \& \text{src1}$	ld[b s l t q]	Load
nand	$\text{dst} := \sim(\text{src2} \& \text{src1})$	st[b s l t q]	Store
or	$\text{dst} := \text{src1} \text{src2}$	READ/MODIFY/WRITE	
nor	$\text{dst} := \sim(\text{src2} \text{src1})$	atadd	Atomic Add (Locked RMW Cycles)
ornot	$\text{dst} := \text{src2} (\sim \text{src1})$	atmod	Atomic Modify (Locked RMW Cycles)
notor	$\text{dst} := (\sim \text{src2}) \text{src1}$	BIT AND BIT FIELD	
xnor	$\text{dst} := (\text{src2} \text{src1}) \& \sim(\text{src2} \& \text{src1})$	setbit	Set a Bit
xor	$\text{dst} := \sim(\text{src2} \text{src1}) (\text{src2} \& \text{src1})$	clrbt	Clear a Bit
not	$\text{dst} := \sim \text{src}$	notbit	Toggle (invert) a Bit
rotate	Rotate Bits	chkbit	Check a Bit and set condition code
BIT AND BIT FIELD		alterbit	Change a Bit according to an operand
setbit	Set a Bit	scanbit	Search src for most significant set bit
clrbt	Clear a Bit	spanbit	Search src for most significant cleared bit
notbit	Toggle (invert) a Bit	extract	Extract specified bit pattern from a word
chkbit	Check a Bit and set condition code	modify	Modify selected bits in dst with src
alterbit	Change a Bit according to an operand	Footnote:	
scanbit	Search src for most significant set bit	i = integer, o = ordinal, b = byte, s = short, w = word (32-bits), l = long, t = triple, q = quad,	
spanbit	Search src for most significant cleared bit	lo = left ordinal, li = left integer, ro = right ordinal, ri = right integer, di = right dividing integer	
extract	Extract specified bit pattern from a word	dst = destination, src = source, x = extended,	
modify	Modify selected bits in dst with src	* = Conditions: lf equal not equal less less or equal greater greater or equal ordered unordered	

Arithmetic shift operations support 32-bit ordinals. Logical shift instructions operate on 32-bit integers, and a 32-bit register value can also be rotated. In addition, all possible two-operand, bitwise Boolean operations exist: AND, NOTAND, ANDNOT, XOR, OR, NOR, XNOR, NOT, NOTOR, ORNOT, and NAND.

Bit and bit field. Bit operations allow bits in the registers to be set, cleared, toggled, and moved to or from the condition codes. SCAN and SPAN operations provide ways to find the most significant set or cleared bit in a register.

The 80960 contains two bit field instructions, EXTRACT and MODIFY. The EXTRACT instruction shifts a bit field in a register to the right and fills in the bits to the left of the bit field with zeros. The MODIFY instruction moves a specified bit field from one register to another when no adjustment change in bit position is required.

Data movement. A set of data movement (MOV) instructions allows register values to be copied to other registers. The MOV instructions can move from one to four registers at once. The load and store operations described later move data to and from memory.

Comparison. These instructions compare operands and set the resulting condition codes in the arithmetic controls register (Figure 3). The 80960's condition codes listed in Table 2 provide the arithmetic flag function of other architectures, in a way that allows maximum parallel execution.

In general, only compare instructions set the 80960's condition codes and conditional instructions use them. To perform an ADD followed by a conditional branch when the result is zero, a Compare and Branch instruction must be executed after the ADD because arithmetic instructions do not alter the condition codes.

Table 2.
Condition code encodings.

Condition code	Condition
000	Unordered *
001	Greater Than
010	Equal (True)
011	Greater Than or Equal
100	Less Than
101	Not Equal (False)
110	Less Than or Equal
111	Ordered

* Used with floating-point data types.

Although not generally noticed in a sequential execution environment, a parallel environment mandates the decoupling of the condition codes from the instruction set. The 80960 allows multiple instructions to execute simultaneously, thus giving ambiguous meaning to a set of condition codes that are altered by multiple arithmetic instructions in the same clock cycle. The 80960 approach separates condition checking and decision making from all other instructions to provide flexibility in reordering instructions for parallel execution.

The 80960 compare instructions compare both integers and ordinals. A subset of the compare instructions increments or decrements an operand after the comparison.

Memory operations

The load/store nature of the architecture decouples memory references from instruction execution. Register set and memory instructions can be executed simultaneously. Since the load data may take some time to

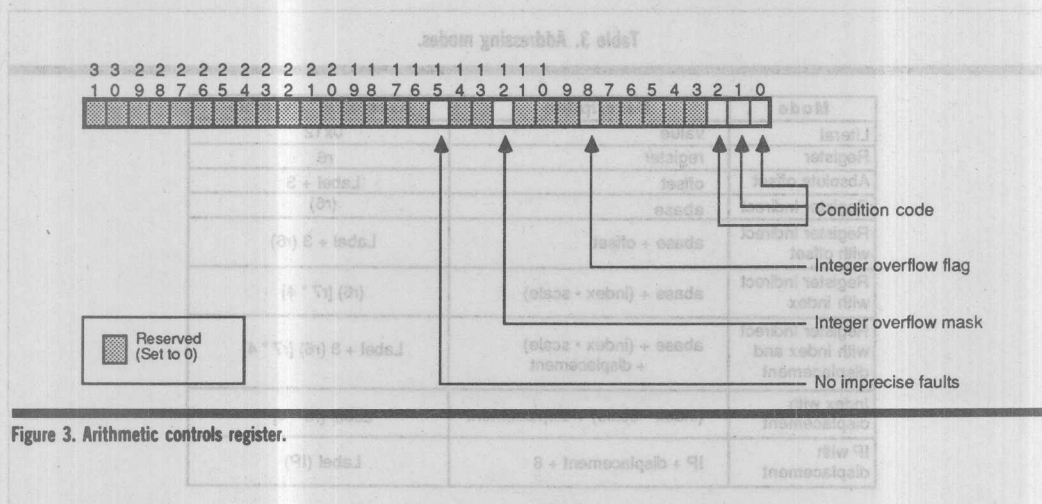


Figure 3. Arithmetic controls register.

80960 architecture

arrive at the CPU, the load requests can be advanced in the instruction stream to overlap memory access time with other data-independent CPU operations.

Load/store. The load and store instructions copy bytes, short words, words, or multiple words to or from memory and registers. When a load integer is specified for an 8-bit or 16-bit quantity, the CPU extends the data's sign to fill 32 bits before writing the destination register. When a load ordinal is specified for an 8-bit or 16-bit quantity, the CPU attaches leading zeros to the data to fill 32 bits before writing the destination register. The store instructions allow the destination data width to be a byte, short word, word, or multiple words. When a store byte, or short word is performed, the CPU automatically formats the data being written according to the data type (integer or ordinal).

Addressing modes. The architecture supports 11 addressing modes for memory operations, as summarized in Table 3. The addressing modes selected for support provide a broad range of most-often-used simple modes. We chose a rich set of addressing modes to allow optimization for code density as well as speed.

Literals are immediate 5-bit numbers that can range from 0 to 31. Literals may be used as operands in any register operation.

The *Register* address mode is used when an operand specifies a register number (g0, r5).

The *Absolute Offset* address mode specifies the absolute address of the target as an offset from the current instruction pointer. The offset is encoded in the memory instruction opcode. If the offset is outside the range of 0 to 2048, the assembler generates a two-word

instruction in which the second word is a 32-bit displacement.

Register Indirect addressing allows the address of the target to be specified by the contents of a register. An immediate offset or displacement can be added to the register to form the effective address. An index (scaled by 2, 4, 8, or 16) may also be added.

Memory operations can also specify target addresses relative to the instruction pointer, a capability useful in creating relocatable data and code.

Atomic memory operations. Two atomic memory operations support multiprocessing environments with more than one processor accessing the same memory; atomic add (ATADD) and atomic modify (ATMOD). The ATADD instruction causes an operand to be added to the value in the specified memory location. The ATMOD causes bits in the specified memory location to be modified under control of a mask. These instructions perform their memory-to-memory, read-modify-write operations with a locked bus to prevent data corruption.

Control operations

Control operations include those instructions that could result in the redirection of program flow. CALL, RET, BRANCH, and COMPARE_AND_BRANCH instructions fall into this category.

Procedure calls. The CALL instruction causes the local registers to be preserved and redirects program flow to a point indicated by an offset encoded in the instruction. The Call Extended (CALLX) instruction dif-

Table 3. Addressing modes.

Mode	Description	Assembler Example
Literal	value	0x12
Register	register	r6
Absolute offset	offset	Label + 3
Register Indirect	abase	(r6)
Register Indirect with offset	abase + offset	Label + 3 (r6)
Register Indirect with index	abase + (index * scale)	(r6) [r7 * 4]
Register Indirect with index and displacement	abase + (index * scale) + displacement	Label + 3 (r6) [r7 * 4]
Index with displacement	(index * scale) + displacement	Label [r6 * 4]
IP with displacement	IP + displacement + 8	Label (IP)

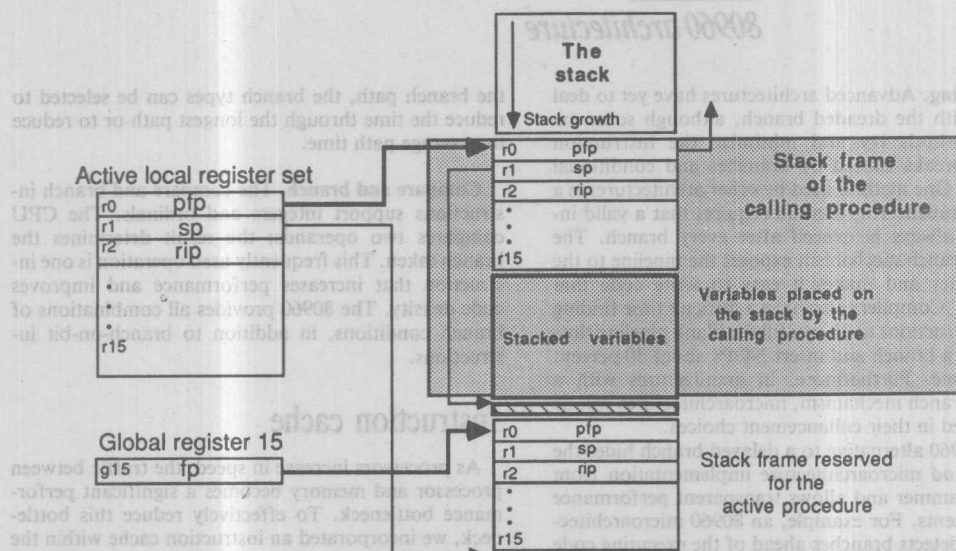


Figure 4. Procedure stack structure.

fers in that it allows a 32-bit value to provide the CALL destination. The destination can either be encoded in the instruction or specified by a register value (for example, indirect call). The Call System (CALLS) instruction gets its target address from a table of system procedure addresses explained later. The Return (RET) instruction returns control to the calling procedure and restores the local registers of the calling procedure.

The semantics of the CALL/RET allow an optimization known as register caching. A register cache keeps the context (local registers) of the most recently executing subroutines on chip so that CALL/RET instructions do not have to access memory to save or restore the local registers.

When a CALL instruction executes, the 80960 allocates a new set of 16 local registers from a pool of register sets for the called procedure. If the pool is depleted, a new register set is allocated by taking one associated with an earlier procedure and saving it in memory. A RET instruction causes the most recently cached local register set to be restored, freeing a register cache location.

The register cache contributes to performance in four ways:

- It significantly reduces the saving and restoring of registers that are usually performed when crossing subroutine boundaries.
- Since the local register sets are mapped into the stack frames, the linkage information that normally appears in stack frames (pointer to previous frame, saved instruction pointer) is contained in the local reg-

isters. Most call and return instructions execute without causing any references to off-chip memory.

- It allows compilers to map most or all of a procedure's local variables directly into registers.
- It provides a large number of registers (16 local and 16 global), which can be exploited by optimizing compilers.

The procedure stack (see Figure 4) reserves space for the cached registers of suspended procedures. When a register set must be flushed from the register cache to memory, it moves to the reserved stack frame space.

When a new procedure is entered, the 80960 allocates space for the procedure's register set as the only contents of its stack frame, although no memory accesses will occur unless the register cache is full. If the procedure desires more space on the stack for autovariables or parameter passing, it adjusts the stack pointer to reserve as much space as it needs. The procedure can then access this space using stack pointer relative addressing so long as the procedure is active. When the procedure returns, its stack is automatically reclaimed.

Branch and Link (BAL) performs a procedure call without saving the local registers. The 80960 saves the return instruction pointer in a global register and redirects program flow. To return from a routine that is invoked by a BAL, a BX (Branch Extended) is performed. BAL allows fast subroutine calls to leaf procedures without allocating (and possibly displacing) a new register set. Since a leaf procedure calls no other procedure, its registers can be allocated out of those remaining in the current set.

80960 architecture

Branching. Advanced architectures have yet to deal cleanly with the dreaded branch, although some existing methods try and minimize the instruction pipeline breaks caused by branches and conditional branches. One method used by other architectures is a delayed branch. This method requires that a valid instruction *always* be placed after every branch. The delayed branch mechanism exposes the pipeline to the programmer and makes it easy to write code that "breaks." Compilers also have a difficult time finding useful instructions to *always* fill the blank pipeline slots following a branch and insert NOPs about 30 percent of the time. Furthermore, in architectures with a delayed branch mechanism, microarchitectures will be constrained in their enhancement choices.

The 80960 alternative to a delayed branch hides the pipeline and microarchitecture implementation from the programmer and allows transparent performance enhancements. For example, an 80960 microarchitecture that detects branches ahead of the executing code could fetch the branch destination to keep the pipeline full. In essence, "branch lookahead" allows branches to be executed in zero clock cycles.

Branches can be unconditional or conditional. The Branch and Branch Extended instructions perform unconditional redirection of program flow without linkage. Branch and Branch Extended differ in the width of the target address offset provided. The Branch instruction includes an encoded offset in the one-word instruction (MEMA format), whereas Branch Extended branches to the location pointed to by a register or an encoded 32-bit displacement (MEMB format).

The conditional branches use the condition codes in the arithmetic controls register to determine whether or not to take the branch. The 80960 provides all combinations of branch conditions.

Branch lookahead works well with unconditional branches but would be of marginal benefit on conditional branches since the branch target, or the instruction after the branch, cannot be executed until after evaluation of the branch condition. Pipeline breaks would, therefore, be inevitable even if the microarchitecture implemented some sort of hardware prediction mechanism. To reduce the effect of the conditional branch on performance, the 80960 defines two types of conditional branches, those that are usually taken and those that aren't usually taken. The implementation can then guess which way the branch is going to go, based upon an excellent resource capable of prediction—the programmer. Only in the case of a programmer's wrong prediction would a pipeline stall occur. Furthermore, compilers will take advantage of branch prediction when they detect loops.

It is either obvious, or uncertain, at the time the program is written which way the branches will branch most often during execution. If the likely branch outcome is obvious, the type of branch to use will be obvious. In the cases where runtime factors determine

the branch path, the branch types can be selected to reduce the time through the longest path or to reduce the average path time.

Compare and branch. The compare and branch instructions support integers and ordinals. The CPU compares two operands; the result determines the branch taken. This frequently used operation is one instruction that increases performance and improves code density. The 80960 provides all combinations of branch conditions, in addition to branch-on-bit instructions.

Instruction cache

As processors increase in speed, the traffic between processor and memory becomes a significant performance bottleneck. To effectively reduce this bottleneck, we incorporated an instruction cache within the processor.

An on-chip instruction cache is highly desirable for two reasons. Caching instructions on chip greatly reduces system bus loading and the criticality of the system's memory access speed in a parallel execution environment. However, an instruction cache plays an additional role. The only way to cause multiple instructions to execute simultaneously is to decode multiple instructions simultaneously. An on-chip instruction cache gives instruction decode the capability of looking downstream and decoding and dispatching multiple instructions simultaneously for parallel execution.

The advantage of an instruction cache over a pre-fetch queue, a technique used in most high-performance microprocessors to date, is that a queue does not reduce the memory traffic for instructions. It only attempts to distribute the traffic more efficiently. A cache's most obvious effect occurs with execution loops, common in embedded control applications. After a loop is first executed, successive iterations of the loop generate no memory references for instruction fetches. Likewise, when a small, low-level procedure concludes and executes a RET instruction, the code for the high-level routine to which it is returning likely still resides in the cache. Thus, we reduce the sensitivity of instruction execution speed to slow memory and free valuable bus bandwidth for other operations.

Having an instruction cache requires special considerations in applications that employ self-modifying code or uploadable programs. In general, embedded applications are unaffected. However, for 80960 chips targeted at embedded applications in which volatile code exists, we will provide implementation-specific cache features. For example, implementations could provide a bus input pin that prohibits the data being read from being cached, a method for flushing the cache, a transparent instruction cache, a cache disable bit, or some other feature tuned to the application.

To allow implementations of the 80960 latitude in the amount and type of cache provided, the architecture does not specify the instruction cache parameterization.

User-supervisor protection

The architecture provides a mode and stack switching mechanism called the user-supervisor protection model. This protection model allows a system design in which the kernel code and data reside in the same address space as the user code and data. However, the access to the kernel procedures (called supervisor procedures) occurs through a specified interface. A data structure called the System Procedure Table provides this interface (Figure 5).

The 80960 references the System Procedure Table when a System Call (CALLS) instruction executes. This call is similar to a local call, except that the processor gets the location of the called procedure from the System Procedure Table. Figure 6 illustrates the use of the CALLS instruction. CALLS requires a procedure-number operand that is used as an index into the table.

The System Procedure Table entry referenced by CALLS specifies an entry pointer and an entry type for the called procedure. The 80960 invokes two types of system procedures, *local* and *supervisor*. A procedure that is specified as a local procedure is invoked as if it were called by the CALL or CALLX instructions, except that the processor gets the entry point of the called procedure from the System Procedure Table. Referencing a supervisor procedure, on the other hand, switches the processor to the supervisor execution mode and to the supervisor stack.

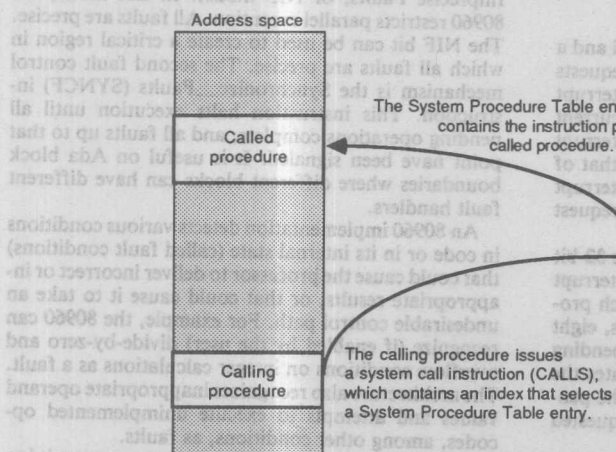


Figure 6. Example of a system procedure call.

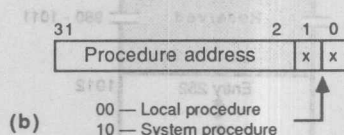
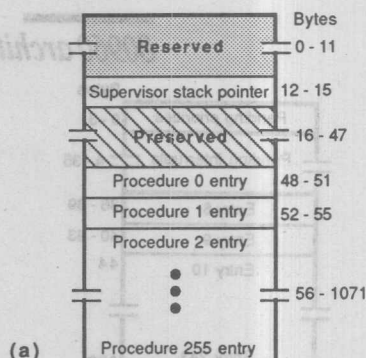


Figure 5. Structure of the System Procedure Table (a) and a procedure entry (b).

Real-time kernel procedures in the supervisor mode execute using a different stack than the one used to execute application programs procedures. Special, supervisor-only instructions also execute in supervisor mode. The MODPC instruction (used to change the processor priority) is always a supervisor instruction. Instruction set extensions that control on-chip hardware are also likely to be restricted to the supervisor mode.

80960 architecture

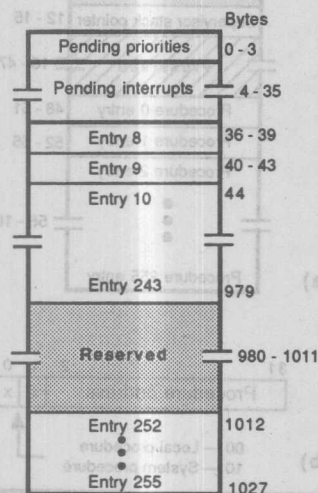


Figure 7. Structure of the interrupt table.

The processor remains in the supervisor mode until the procedure that caused the original mode switch performs a return. Switching stacks and protecting against stack corruption help maintain the integrity of the kernel. For example, the mechanism allows access to system debugging software or a system monitor even if the application crashes.

Interrupts

The 80960 contains a priority interrupt model and a mechanism for queuing pending interrupt requests without user program intervention. When an interrupt is signaled and its priority is higher than the current processor priority, the CPU invokes an interrupt handler and the processor priority changes to that of the interrupt. Otherwise, the 80960 saves the interrupt for service until it becomes the highest priority request pending.

The interrupt table seen in Figure 7 holds the 32-bit pending priorities field, the 256-bit pending interrupt field, and the 248 interrupt vectors. Within each processor priority the 80960 contains eight vectors, eight associated pending interrupt bits, and one pending priority bit. The pending priorities field indicates the priorities at which pending interrupts await. The pending interrupt field indicates exactly which requested interrupts have not yet been serviced.

A pending priority bit is simply the OR of all eight pending interrupt bits at a particular priority. This field optimizes checking for pending interrupts by the processor. When an interrupt request will not be serviced

immediately, the 80960 sets the bit in the pending interrupt field associated with the request. It also sets the pending priorities bit associated with the priority of the request. When the running priority of the processor drops below that of the pending interrupt, the 80960 services the interrupt and clears the associated pending bit. The CPU also clears the associated pending priority bit if appropriate.

Faults

Processors use fault mechanisms to handle exceptions or errant conditions that a program may or may not be capable of correcting. We defined the 80960's fault mechanism for an environment in which parallel or out-of-order execution occurs. When a fault is generated, the processor calls the appropriate fault handler. The 80960 automatically provides the handler with an extensive set of information about the faulting condition for correction or analysis.

It is possible that when a fault is detected not enough information would exist to determine the exact instruction that faulted. For example, when multiple instructions execute in one clock cycle, multiple faults could occur in a single clock cycle. This "imprecise" condition could generate a fault that we call imprecise. A tightly coupled fault handler may be able to recover proper program execution when an imprecise fault occurs. Precise faults are those from which recovery is easy.

The 80960 provides two controls over the generation of imprecise conditions and faults. The first fault control method, a global control bit, puts the processor in a mode where no imprecise conditions are created (No Imprecise Faults, or NIF mode). In this mode, the 80960 restricts parallel execution. All faults are precise. The NIF bit can be used to create a critical region in which all faults are precise. The second fault control mechanism is the Synchronize_Faults (SYNCF) instruction. This instruction halts execution until all pending operations complete, and all faults up to that point have been signaled. It is useful on Ada block boundaries where different blocks can have different fault handlers.

An 80960 implementation detects various conditions in code or in its internal state (called fault conditions) that could cause the processor to deliver incorrect or inappropriate results, or that could cause it to take an undesirable control path. For example, the 80960 can recognize (if enabled by the user) divide-by-zero and overflow conditions on integer calculations as a fault. The architecture also recognizes inappropriate operand values and attempts to execute unimplemented opcodes, among other conditions, as faults.

When a fault is detected, the system processes it immediately and independently of the program or handler that is executing at the time. The fault mechanism is similar to that used by the interrupts. Several fault

types exist, in which the fault type determines which entry in the Fault Table (Figure 8) is invoked for a particular fault. The Fault Table contains one entry for each fault type. The entry defines a particular fault handler routine as a local procedure or a system procedure. When the fault handler is a local procedure, the Fault Table entry contains the address of the procedure entry point. When the fault handler is a system procedure, the Fault Table entry contains the system procedure number, which selects the correct entry point from the System Procedure Table described earlier.

Figure 9 describes the fault record, which is the information provided to a fault handler when a fault occurs. Table 4 on page 76 summarizes the fault types

and subtypes that are currently defined in the 80960 architecture. As extensions to the architecture that consume additional fault types become available, the encoding of fault types and subtypes will occur in such a way that every implementation capable of recognizing similar faulting conditions encodes them identically. For example, the 80960KB adds the floating-point faults (fault type 4). Any other 80960 implementations that also recognize floating-point faults also encode them as fault type 4.

Debug support

Another objective of the architecture is to support software debugging and tracing. A trace-controls register enables most of this support. The trace controls detect any combination of the following events:

- Instruction execution (single step),
- Execution of a Taken Branch instruction,
- Execution of a Call instruction,
- Execution of a Return instruction,

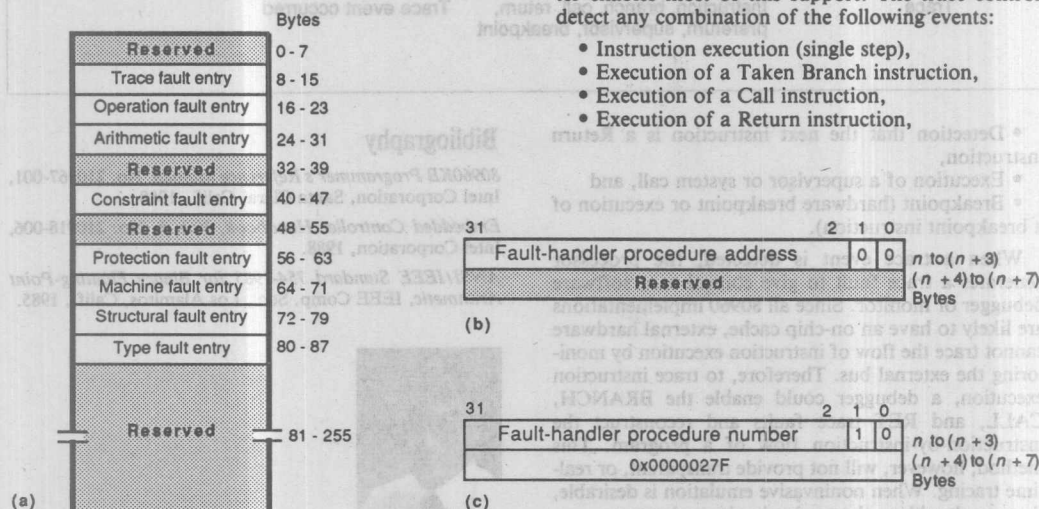


Figure 8. Structure of the fault table (a); an entry to reference a local procedure (b); and an entry to reference a system procedure (c).

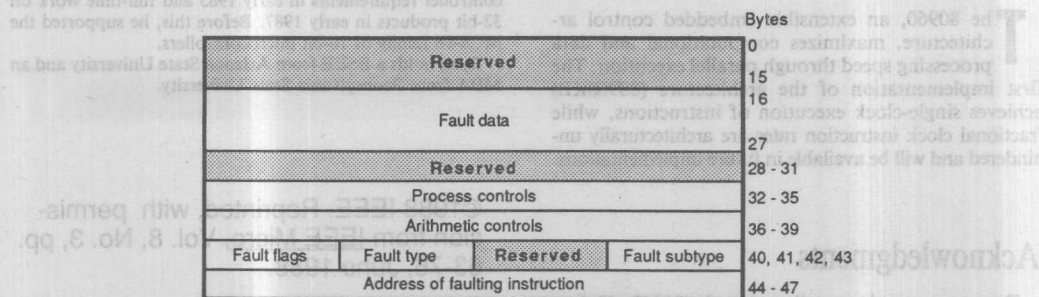


Figure 9. Fault record information. The return pointer r2 is also provided.

80960 architecture

Table 4. Fault types and subtypes.

Fault type	Fault Subtypes	Comments
Arithmetic	Overflow, underflow	Integer overflows/ divide by zero
Constraint	Range	If FAULT_IF taken
Protection	Length	Procedure # in CALLS out of range
Machine	Bad access	Memory access failed to complete
Type	Mismatch	Tried to execute supervisor instruction in nonsupervisor mode
Operation	Invalid opcode, Invalid operand	Tried to execute invalid opcode, or an operand in a valid opcode was invalid
Trace	Instruction, branch, call, return, prereturn, supervisor, breakpoint	Trace event occurred

- Detection that the next instruction is a Return instruction,
- Execution of a supervisor or system call, and
- Breakpoint (hardware breakpoint or execution of a breakpoint instruction).

When a trace event is detected, the processor generates a trace fault to give control to a software debugger or monitor. Since all 80960 implementations are likely to have an on-chip cache, external hardware cannot trace the flow of instruction execution by monitoring the external bus. Therefore, to trace instruction execution, a debugger could enable the BRANCH, CALL, and RET trace faults and reconstruct the instruction-by-instruction flow of a program. This method, however, will not provide transparent, or real-time tracing. When noninvasive emulation is desirable, the user should employ an in-circuit emulator.

The 80960, an extensible embedded control architecture, maximizes computational and data processing speed through parallel execution. The first implementation of the architecture (80960KB) achieves single-clock execution of instructions, while fractional clock instruction rates are architecturally unhindered and will be available in future implementations.

Acknowledgments

Too many people contributed to the 80960 effort to list them here. However, I relied upon the following, either in person or through their writings, in developing this article: Dave Budde, Glen Hinton, Mike Imel, Konrad Lai, Glenford J. Myers, Lew Pacely, Fred Pollack, Rob Riches, Frank Smith, and Randy Steck.

Bibliography

- 80960KB Programmer's Reference Manual, No. 210567-001, Intel Corporation, Santa Clara, Calif., 1988.
- Embedded Controller Handbook, Vol. 1, No. 210918-006, Intel Corporation, 1988.
- ANSI/IEEE Standard 754-1985 for Binary Floating-Point Arithmetic, IEEE Comp. Soc., Los Alamitos, Calif., 1985.



David P. Ryan, a senior applications engineer in Intel's embedded controller operation, supervises the Arizona-based 32-bit applications team. He began work on 32-bit embedded controller requirements in early 1985 and full-time work on 32-bit products in early 1987. Before this, he supported the MCS-96 family of 16-bit microcontrollers.

Ryan holds a BSEE from Arizona State University and an MBA from Pennsylvania State University.

©1988 IEEE. Reprinted, with permission from *IEEE Micro*, Vol. 8, No. 3, pp. 63-76, June 1988.

General Microcontroller Application Notes

9

Digital circuits are often thought of as being immune to noise problems, but really they're not. Noises in digital systems produce software upsets: program jumps to apparently random locations in memory. Noise-induced glitches in the signal lines can cause such problems, but the supply voltage is more sensitive to glitches than the signal lines.

Severe noise conditions, those involving electrostatic discharges, or as found in automotive environments, can do permanent damage to the hardware. Electrostatic discharges can blow a crater in the silicon. In the automotive environment, in ordinary operation, the "12V" power line can show + and -400V transients.

This Application Note describes some electrical noises and noise environments. Design considerations, along the lines of PCB layout, power supply distribution and decoupling, and shielding and grounding techniques, that may help minimize noise susceptibility are reviewed. Special attention is given to the automotive and ESD environments.

Symptoms of Noise Problems

Noise problems are not usually encountered during the development phase of a microcontroller system. This is because benches rarely simulate the system's intended environment. Noise problems tend not to show up until the system is installed and operating in its intended environment. Then, after a few minutes or hours of normal operation the system finds itself someplace out in left field. Inputs are ignored and outputs are gibberish. The system may respond to a reset, or it may have to be turned off physically and then back on again, at which point it commences operating as though nothing had happened. There may be an obvious cause, such as an electrostatic discharge from somebody's finger to a keyboard or the upset occurs every time a copier machine is turned on or off. Or there may be no obvious cause, and nothing the operator can do will make the upset repeat itself. But a few minutes, or a few hours, or a few days later it happens again.

One symptom of electrical noise problems is randomness, both in the occurrence of the problem and in what the system does in its failure. All operational upsets that occur at seemingly random intervals are not necessarily caused by noise in the system. Marginal VCC, inadequate decoupling, rarely encountered software conditions, or timing coincidences can produce upsets that seem to occur randomly. On the other hand, some noise sources can produce upsets downright periodically. Nevertheless, the more difficult it is to characterize an upset as to cause and effect, the more likely it is to be a noise problem.

Types and Sources of Electrical Noise

The name given to electrical noises other than those that are inherent in the circuit components (such as thermal noise) is EMI: electromagnetic interference. Motors, power switches, fluorescent lights, electrostatic discharges, etc., are sources of EMI. There is a veritable alphabet soup of EMI types, and these are briefly described below.

SUPPLY LINE TRANSIENTS

Anything that switches heavy current loads onto or off of AC or DC power lines will cause large transients in these power lines. Switching an electric typewriter on or off, for example, can put a 1000V spike onto the AC power lines.

The basic mechanism behind supply line transients is shown in Figure 1. The battery represents any power source, AC or DC. The coils represent the line inductance between the power source and the switchable loads R1 and R2. If both loads are drawing current, the line current flowing through the line inductance establishes a magnetic field of some value. Then, when one of the loads is switched off, the field due to that component of the line current collapses, generating transient voltages, $v = L(di/dt)$, which try to maintain the current at its original level. That's called an "inductive kick." Because of contact bounce, transients are generated whether the switch is being opened or closed, but they're worse when the switch is being opened.

An inductive kick of one type or another is involved in most line transients, including those found in the automotive environment. Other mechanisms for line transients exist, involving noise pickup on the lines. The noise voltages are then conducted to a susceptible circuit right along with the power.

EMP AND RFI

Anything that produces arcs or sparks will radiate electromagnetic pulses (EMP) or radio-frequency interference (RFI).

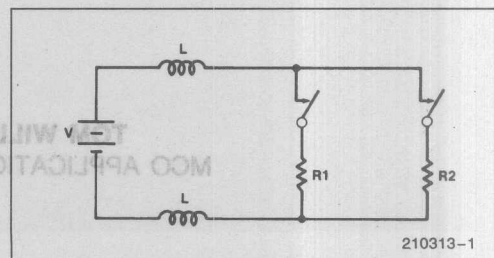


Figure 1. Supply Line Transients

Spark discharges have probably caused more software upsets in digital equipment than any other single noise source. The upsetting mechanism is the EMP produced by the spark. The EMP induces transients in the circuit, which are what actually cause the upset.

Arcs and sparks occur in automotive ignition systems, electric motors, switches, static discharges, etc. Electric motors that have commutator bars produce an arc as the brushes pass from one bar to the next. DC motors and the "universal" (AC/DC) motors that are used to power hand tools are the kinds that have commutator bars. In switches, the same inductive kick that puts transients on the supply lines will cause an opening or closing switch to throw a spark.

ESD

Electrostatic discharge (ESD) is the spark that occurs when a person picks up a static charge from walking across a carpet, and then discharges it into a keyboard, or whatever else can be touched. Walking across a carpet in a dry climate, a person can accumulate a static voltage of 35kV. The current pulse from an electrostatic discharge has an extremely fast risetime — typically, 4A/ns. Figure 2 shows ESD waveforms that have been observed by some investigators of ESD phenomena.

It is enlightening to calculate the $L(di/dt)$ voltage required to drive an ESD current pulse through a couple of inches of straight wire. Two inches of straight wire has about 50 nH of inductance. That's not very much, but using 50 nH for L and 4A/ns for di/dt gives an $L(di/dt)$ drop of about 200V. Recent observations by W.M. King suggest even faster risetimes (Figure 2b) and the occurrence of multiple discharges during a single discharge event.

Obviously, ESD-sensitivity needs to be considered in the design of equipment that is going to be subjected to it, such as office equipment.

GROUND NOISE

Currents in ground lines are another source of noise. These can be 60 Hz currents from the power lines, or RF hash, or crosstalk from other signals that are sharing this particular wire as a signal return line. Noise in the ground lines is often referred to as a "ground loop" problem. The basic concept of the ground loop is shown in Figure 3. The problem is that true earth-ground is not really at the same potential in all locations. If the two ends of a wire are earth-grounded at different locations, the voltage difference between the two "ground" points can drive significant currents (several amperes) through the wire. Consider the wire to be part of a loop which contains, in addition to the wire, a voltage source that represents the difference in potential between the two ground points, and you have

the classical "ground loop." By extension, the term is used to refer to any unwanted (and often unexpected) currents in a ground line.

"Radiated" and "Conducted" Noise

Radiated noise is noise that arrives at the victim circuit in the form of electromagnetic radiation, such as EMP and RFI. It causes trouble by inducing extraneous voltages in the circuit. Conducted noise is noise that arrives at the victim circuit already in the form of an extraneous voltage, typically via the AC or DC power lines.

One defends against radiated noise by care in designing layouts and the use of effective shielding techniques. One defends against conducted noise with filters and

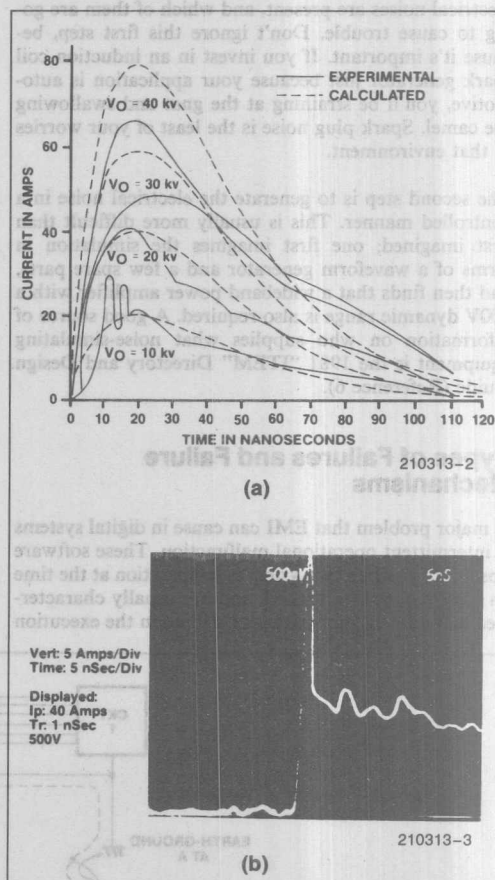


Figure 2. Waveforms of Electrostatic Discharge Currents From a Hand-Held Metallic Object

suppressors, although layouts and grounding techniques are important here, too.

Simulating the Environment

Addressing noise problems after the design of a system has been completed is an expensive proposition. The ill will generated by failures in the field is not cheap either. It's cheaper in the long run to invest a little time and money in learning about noise and noise simulation equipment, so that controlled tests can be made on the bench as the design is developing.

Simulating the intended noise environment is a two-step process: First you have to recognize what the noise environment is, that is, you have to know what kinds of electrical noises are present, and which of them are going to cause trouble. Don't ignore this first step, because it's important. If you invest in an induction coil spark generator just because your application is automotive, you'll be straining at the gnat and swallowing the camel. Spark plug noise is the least of your worries in that environment.

The second step is to generate the electrical noise in a controlled manner. This is usually more difficult than first imagined; one first imagines the simulation in terms of a waveform generator and a few spare parts, and then finds that a wideband power amplifier with a 200V dynamic range is also required. A good source of information on who supplies what noise-simulating equipment is the 1981 "ITEM" Directory and Design Guide (Reference 6).

Types of Failures and Failure Mechanisms

A major problem that EMI can cause in digital systems is intermittent operational malfunction. These software upsets occur when the system is in operation at the time an EMI source is activated, and are usually characterized by a loss of information or a jump in the execution

of the program to some random location in memory. The person who has to iron out such problems is tempted to say the program counter went crazy. There is usually no damage to the hardware, and normal operation can resume as soon as the EMI has passed or the source is de-activated. Resuming normal operation usually requires manual or automatic reset, and possibly re-entering of lost information.

Electrostatic discharges from operating personnel can cause not only software upsets, but also permanent ("hard") damage to the system. For this to happen the system doesn't even have to be in operation. Sometimes the permanent damage is latent, meaning the initial damage may be marginal and require further aggravation through operating stress and time before permanent failure takes place. Sometimes too the damage is hidden.

One ESD-related failure mechanism that has been identified has to do with the bias voltage on the substrate of the chip. On some CPU chips the substrate is held at $-2.5V$ by a phase-shift oscillator working into a capacitor/diode clamping circuit. This is called a "charge pump" in chip-design circles. If the substrate wanders too far in either direction, program read errors are noted. Some designs have been known to allow electrostatic discharge currents to flow directly into port pins of an 8048. The resulting damage to the oxide causes an increase in leakage current, which loads down the charge pump, reducing the substrate voltage to a marginal or unacceptable level. The system is then unreliable or completely inoperative until the CPU chip is replaced. But if the CPU chip was subjected to a discharge spark once, it will eventually happen again.

Chips that have a grounded substrate, such as the 8748, can sometimes sustain some oxide damage without actually becoming inoperative. In this case the damage is present, and the increased leakage current is noted; however, since the substrate voltage retains its design value, the damage is largely hidden.

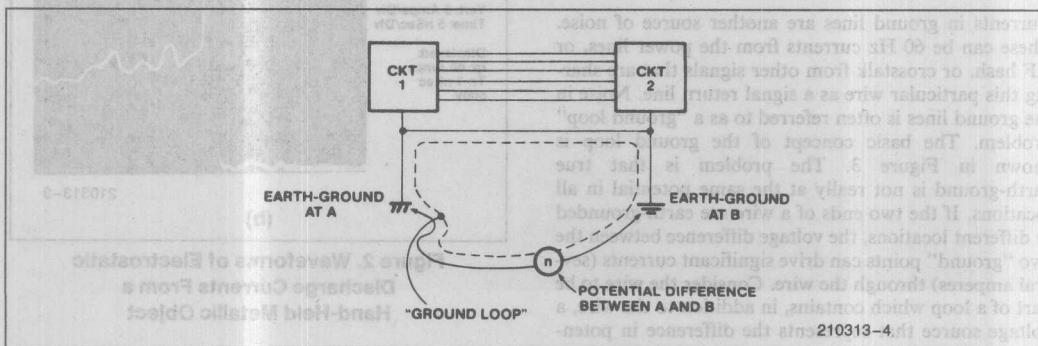


Figure 3. What a Ground Loop Is

It must therefore be recognized that connecting port pins unprotected to a keyboard or to anything else that is subject to electrostatic discharges, makes an extremely dangerous configuration. It doesn't make any difference what CPU chip is being used, or who makes it. If it connects unprotected to a keyboard, it will eventually be destroyed. Designing for an ESD-environment will be discussed further on.

We might note here that MOS chips are not the only components that are susceptible to permanent ESD damage. Bipolar and linear chips can also be damaged in this way. PN junctions are subject to a hard failure mechanism called thermal secondary breakdown, in which a current spike, such as from an electrostatic discharge, causes microscopically localized spots in the junction to approach melt temperatures. Low power TTL chips are subject to this type of damage, as are op-amps. Op-amps, in addition, often carry on-chip MOS capacitors which are directly across an external pin combination, and these are susceptible to dielectric breakdown.

We return now to the subject of software upsets. Noise transients can upset the chip through any pin, even an output pin, because every pin on the chip connects to the substrate through a pn junction. However, the most vulnerable pin is probably the VCC line, since it has direct access to all parts of the chip: every register, gate, flip-flop and buffer.

The menu of possible upset mechanisms is quite lengthy. A transient on the substrate at the wrong time will generally cause a program read error. A false level at a control input can cause an extraneous or misdirected opcode fetch. A disturbance on the supply line can flip a bit in the program counter or instruction register. A short interruption or reversal of polarity on the supply line can actually turn the processor off, but not long enough for the power-up reset capacitor to discharge. Thus when the transient ends, the chip starts up again without a reset.

A common failure mode is for the processor to lock itself into a tight loop. Here it may be executing the data in a table, or the program counter may have jumped a notch, so that the processor is now executing operands instead of opcodes, or it may be trying to fetch opcodes from a nonexistent external program memory.

It should be emphasized that mechanisms for upsets have to do with the arrival of noise-induced transients at the pins of the chips, rather than with the generation of noise pulses within the chip itself, that is, it's not the chip that is picking up noise, it's the circuit.

The Game Plan

Prevention is usually cheaper than suppression, so first we'll consider some preventive methods that might help

to minimize the generation of noise voltages in the circuit. These methods involve grounding, shielding, and wiring techniques that are directed toward the mechanisms by which noise voltages are generated in the circuit. We'll also discuss methods of decoupling. Then we'll look at some schemes for making a graceful recovery from upsets that occur in spite of preventive measures. Lastly, we'll take another look at two special problem areas: electrostatic discharges and the automotive environment.

Current Loops

The first thing most people learn about electricity is that current won't flow unless it can flow in a closed loop. This simple fact is sometimes temporarily forgotten by the overworked engineer who has spent the past several years mastering the intricacies of the DO loop, the timing loop, the feedback loop, and maybe even the ground loop. The simple current loop probably owes its apparent demise to the invention of the ground symbol. By a stroke of the pen one avoids having to draw the return paths of most of the current loops in the circuit. Then "ground" turns into an infinite current sink, so that any current that flows into it is gone and forgotten. Forgotten it may be, but it's not gone. It must return to its source, so that its path will by all the laws of nature form a closed loop.

The physical geometry of a given current loop is the key to why it generates EMI, why it's susceptible to EMI, and how to shield it. Specifically, it's the area of the loop that matters.

Any flow of current generates a magnetic field whose intensity varies inversely to the distance from the wire that carries the current. Two parallel wires conducting currents $+I$ and $-I$ (as in signal feed and return lines) would generate a nonzero magnetic field near the wires, where the distance from a given point to one wire is noticeably different from the distance to the other wire, but farther away (relative to the wire spacing), where the distances from a given point to either wire are about the same, the fields from both wires tend to cancel out. Thus, maintaining proximity between feed and return paths is an important way to minimize their interference with other signals. The way to maintain their proximity is essentially to minimize their loop area. And, because the mutual inductance from current loop A to current loop B is the same as the mutual inductance from current loop B to current loop A, a circuit that doesn't radiate interference doesn't receive it either.

Thus, from the standpoint of reducing both generation of EMI and susceptibility to EMI, the hard rule is to keep loop areas small. To say that loop areas should be minimized is the same as saying the circuit inductance

should be minimized. Inductance is by definition the constant of proportionality between current and the magnetic field it produces: $\phi = LI$. Holding the feed and return wires close together so as to promote field cancellation can be described either as minimizing the loop area or as minimizing L . It's the same thing.

Shielding

There are three basic kinds of shields: shielding against capacitive coupling, shielding against inductive coupling, and RF shielding. Capacitive coupling is electric field coupling, so shielding against it amounts to shielding against electric fields. As will be seen, this is relatively easy. Inductive coupling is magnetic field coupling, so shielding against it is shielding against magnetic fields. This is a little more difficult. Strangely enough, this type of shielding does not in general involve the use of magnetic materials. RF shielding, the classical "metallic barrier" against all sorts of electromagnetic fields, is what most people picture when they think about shielding. Its effectiveness depends partly on the selection of the shielding material, but mostly, as it turns out, on the treatment of its seams and the geometry of its openings.

SHIELDING AGAINST CAPACITIVE COUPLING

Capacitive coupling involves the passage of interfering signals through mutual or stray capacitances that aren't shown on the circuit diagram, but which the experienced engineer knows are there. Capacitive coupling to one's body is what would cause an unstable oscillator to change its frequency when the person reaches his hand over the circuit, for example. More importantly, in a digital system it causes crosstalk in multi-wire cables.

The way to block capacitive coupling is to enclose the circuit or conductor you want to protect in a metal shield. That's called an electrostatic or Faraday shield. If coverage is 100%, the shield does not have to be grounded, but it usually is, to ensure that circuit-to-shield capacitances go to signal reference ground rather than act as feedback and crosstalk elements. Besides, from a mechanical point of view, grounding it is almost inevitable.

A grounded Faraday shield can be used to break capacitive coupling between a noisy circuit and a victim circuit, as shown in Figure 4. Figure 4a shows two circuits capacitively coupled through the stray capacitance between them. In Figure 4b the stray capacitance is intercepted by a grounded Faraday shield, so that interference currents are shunted to ground. For example, a grounded plane can be inserted between PCBs (printed circuit boards) to eliminate most of the capacitive coupling between them.

Another application of the Faraday shield is in the electrostatically shielded transformer. Here, a conducting foil is laid between the primary and secondary coils so as to intercept the capacitive coupling between them. If a system is being upset by AC line transients, this type of transformer may provide the fix. To be effective in this application, the shield must be connected to the greenwire ground.

SHIELDING AGAINST INDUCTIVE COUPLING

With inductive coupling, the physical mechanism involved is a magnetic flux density B from some external interference source that links with a current loop in the victim circuit, and generates a voltage in the loop in accordance with Lenz's law: $v = -NA(dB/dt)$, where in this case $N = 1$ and A is the area of the current loop in the victim circuit.

There are two aspects to defending a circuit against inductive pickup. One aspect is to try to minimize the offensive fields at their source. This is done by minimizing the area of the current loop at the source so as to promote field cancellation, as described in the section on current loops. The other aspect is to minimize the inductive pickup in the victim circuit by minimizing the area of that current loop, since, from Lenz's law, the induced voltage is proportional to this area. So the two aspects really involve the same corrective action: minimize the areas of the current loops. In other words, minimizing the offensiveness of a circuit inherently minimizes its susceptibility.

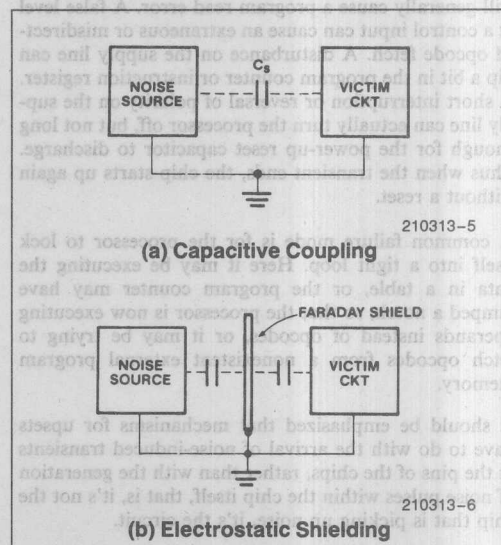


Figure 4. Use of Faraday Shield

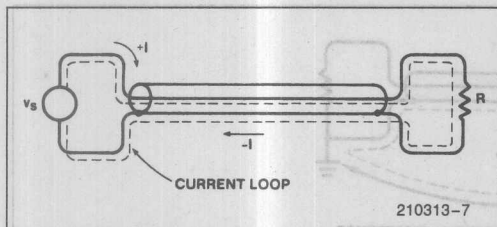


Figure 5. External to the Shield, $\phi = 0$

Shielding against inductive coupling means nothing more nor less than controlling the dimensions of the current loops in the circuit. We must look at four examples of this type of “shielding”: the coaxial cable, the twisted pair, the ground plane, and the gridded-ground PCB layout.

The Coaxial Cable—Figure 5 shows a coaxial cable carrying a current I from a signal source to a receiving load. The shield carries the same current as the center conductor. Outside the shield, the magnetic field produced by $+I$ flowing in the center conductor is cancelled by the field produced by $-I$ flowing in the shield. To the extent that the cable is ideal in producing zero external magnetic field, it is immune to inductive pickup from external sources. The cable adds effectively zero area to the loop. This is true only if the shield carries the same current as the center conductor.

In the real world, both the signal source and the receiving load are likely to have one end connected to a common signal ground. In that case, should the cable be grounded at one end, both ends, or neither end? The answer is that it should be grounded at both ends. Figure 6a shows the situation when the cable shield is grounded at only one end. In that case the current loop runs down the center conductor of the cable, then back through the common ground connection. The loop area is not well defined. The shield not only does not carry the same current as the center conductor, but it doesn't carry any current at all. There is no field cancellation at all. The shield has no effect whatsoever on either the generation of EMI or susceptibility to EMI. (It is, however, still effective as an electrostatic shield, or at least it would be if the shield coverage were 100%.)

Figure 6b shows the situation when the cable is grounded at both ends. Does the shield carry all of the return current, or only a portion of it on account of the shunting effect of the common ground connection? The answer to that question depends on the frequency content of the signal. In general, the current loop will follow the path of least impedance. At low frequencies, 0 Hz to several kHz, where the inductive reactance is insignificant, the current will follow the path of least resistance. Above a few kHz, where inductive reactance predominates, the current will follow the path of least inductance. The path of least inductance is the path of

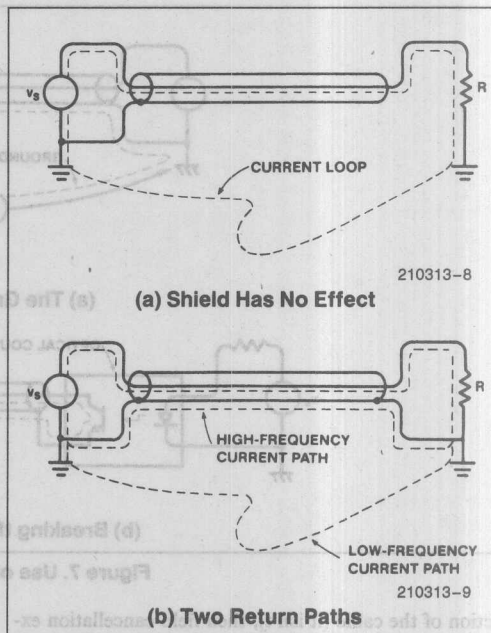


Figure 6. Use of Coaxial Cable

minimum loop area. Hence, for higher frequencies the shield carries virtually the same current as the center conductor, and is therefore effective against both generation and reception of EMI.

Note that we have now introduced the famous “ground loop” problem, as shown in Figure 7a. Fortunately, a digital system has some built-in immunity to moderate ground loop noise. In a noisy environment, however, one can break the ground loop, and still maintain the shielding effectiveness of the coaxial cable, by inserting an optical coupler, as shown in Figure 7b. What the optical coupler does, basically, is allow us to re-define the signal source as being ungrounded, so that that end of the cable need not be grounded, and still lets the shield carry the same current as the center conductor. Obviously, if the signal source weren't grounded in the first place, the optical coupler wouldn't be needed.

The Twisted Pair—A cheaper way to minimize loop area is to run the feed and return wires right next to each other. This isn't as effective as a coaxial cable in minimizing loop area. An ideal coaxial cable adds zero area to the loop, whereas merely keeping the feed and return wires next to each other is bound to add a finite area.

However, two things work to make this cheaper method almost as good as a coaxial cable. First, real coaxial cables are not ideal. If the shield current isn't evenly distributed around the center conductor at every cross-

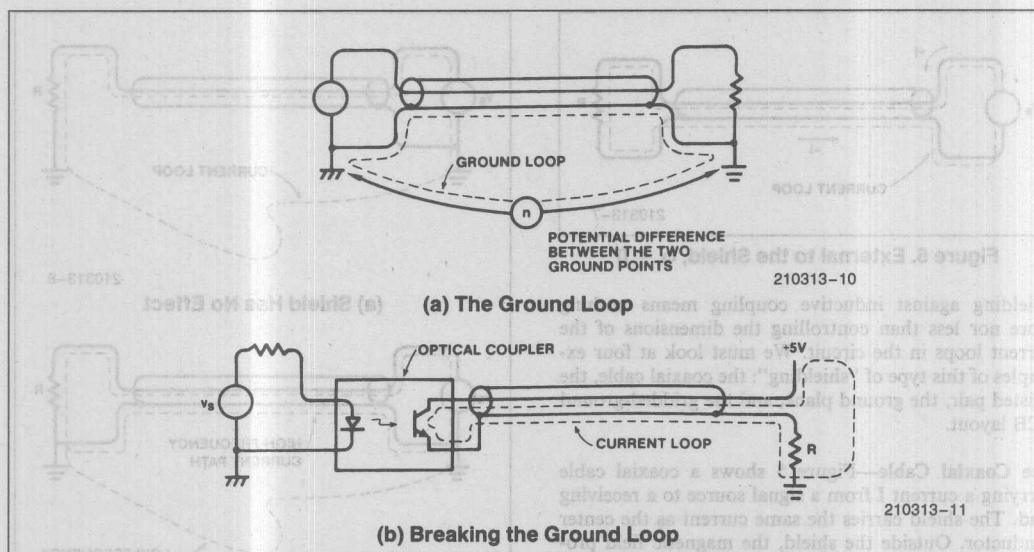


Figure 7. Use of Optical Coupler

section of the cable (it isn't), then field cancellation external to the shield is incomplete. If field cancellation is incomplete, then the effective area added to the loop by the cable isn't zero. Second, in the cheaper method the feed and return wires can be twisted together. This not only maintains their proximity, but the noise picked up in one twist tends to cancel out the noise picked up in the next twist down the line. Thus the "twisted pair" turns out to be about as good a shield against inductive coupling as coaxial cable is.

The twisted pair does not, however, provide electrostatic shielding (i.e., shielding against capacitive coupling). Another operational difference between them is that the coaxial cable works better at higher frequencies. This is primarily because the twisted pair adds more capacitive loading to the signal source than the coaxial cable does. The twisted pair is normally considered useful up to only about 1 MHz, as opposed to near a GHz for the coaxial cable.

The Ground Plane—The best way to minimize loop areas when many current loops are involved is to use a ground plane. A ground plane is a conducting surface that is to serve as a return conductor for all the current loops in the circuit. Normally, it would be one or more layers of a multilayer PCB. All ground points in the circuit go not to a grounded trace on the PCB, but directly to the ground plane. This leaves each current loop in the circuit free to complete itself in whatever configuration yields minimum loop area (for frequencies wherein the ground path impedance is primarily inductive).

Thus, if the feed path for a given signal zigzags its way across the PCB, the return path for this signal is free to zigzag right along beneath it on the ground plane, in such a configuration as to minimize the energy stored in the magnetic field produced by this current loop. Minimal magnetic flux means minimal effective loop area and minimal susceptibility to inductive coupling.

The Gridded-Ground PCB Layout—The next best thing to a ground plane is to lay out the ground traces on a PCB in the form of a grid structure, as shown in Figure 8. Laying horizontal traces on one side of the board and vertical traces on the other side allows the passage of signal and power traces. Wherever vertical and horizontal ground traces cross, they must be connected by a feed-through.

Have we not created here a network of "ground loops"? Yes, in the literal sense of the word, but loops in the ground layout on a PCB are not to be feared. Such inoffensive little loops have never caused as much noise pickup as their avoidance has. Trying to avoid innocent little loops in the ground layout, PCB designers have forced current loops into geometries that could swallow a whale. That is exactly the wrong thing to do.

The gridded ground structure works almost as well as the ground plane, as far as minimizing loop area is concerned. For a given current loop, the primary return path may have to zig once in a while where its feed path zags, but you still get a mathematically optimal dis-

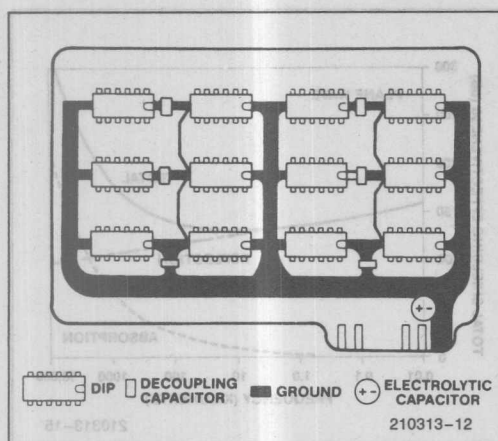


Figure 8. PCB with Gridded Ground

tribution of currents in the grid structure, such that the current loop produces less magnetic flux than if the return path were restrained to follow any single given ground trace. The key to attaining minimum loop areas for all the current loops together is to let the ground currents distribute themselves around the entire area of the board as freely as possible. They want to minimize their own magnetic field. Just let them.

RF SHIELDING

A time-varying electric field generates a time-varying magnetic field, and vice versa. Far from the source of a time-varying EM field, the ratio of the amplitudes of the electric and magnetic fields is always 377Ω . Up close to the source of the fields, however, this ratio can be quite different, and dependent on the nature of the source. Where the ratio is near 377Ω is called the far field, and where the ratio is significantly different from 377Ω is called the near field. The ratio itself is called the wave impedance, E/H.

The near field goes out about $1/6$ of a wavelength from the source. At 1 MHz this is about 150 feet, and at 10 MHz it's about 15 feet. That means if an EMI source is in the same room with the victim circuit, it's likely to be a near field problem. The reason this matters is that in the near field an RF interference problem could be almost entirely due to E-field coupling or H-field coupling, and that could influence the choice of an RF shield or whether an RF shield will help at all.

In the near field of a whip antenna, the E/H ratio is higher than 377Ω , which means it's mainly an E-field generator. A wire-wrap post can be a whip antenna. Interference from a whip antenna would be by electric field coupling, which is basically capacitive coupling. Methods to protect a circuit from capacitive coupling, such as a Faraday shield, would be effective

against RF interference from a whip antenna. A gridded-ground structure would be less effective.

In the near field of a loop antenna, the E/H ratio is lower than 377Ω , which means it's mainly an H-field generator. Any current loop is a loop antenna. Interference from a loop antenna would be by magnetic field coupling, which is basically the same as inductive coupling. Methods to protect a circuit from inductive coupling, such as a gridded-ground structure, would be effective against RF interference from a loop antenna. A Faraday shield would be less effective.

A more difficult case of RF interference, near field or far field, may require a genuine metallic RF shield. The idea behind RF shielding is that time-varying EMI fields induce currents in the shielding material. The induced currents dissipate energy in two ways: I^2R losses in the shielding material and radiation losses as they re-radiate their own EM fields. The energy for both of these mechanisms is drawn from the impinging EMI fields. Hence the EMI is weakened as it penetrates the shield.

More formally, the I^2R losses are referred to as absorption loss, and the re-radiation is called reflection loss. As it turns out, absorption loss is the primary shielding mechanism for H-fields, and reflection loss is the primary shielding mechanism for E-fields. Reflection loss, being a surface phenomenon, is pretty much independent of the thickness of the shielding material. Both loss mechanisms, however, are dependent on the frequency (ω) of the impinging EMI field, and on the permeability (μ) and conductivity (σ) of the shielding material. These loss mechanisms vary approximately as follows:

$$\text{reflection loss to an E-field (in dB)} \sim \log \frac{\sigma}{\omega \mu}$$

$$\text{absorption loss to an H-field (in dB)} \sim t \sqrt{\omega \sigma \mu}$$

where t is the thickness of the shielding material.

The first expression indicates that E-field shielding is more effective if the shield material is highly conductive, and less effective if the shield is ferromagnetic, and that low-frequency fields are easier to block than high-frequency fields. This is shown in Figure 9.

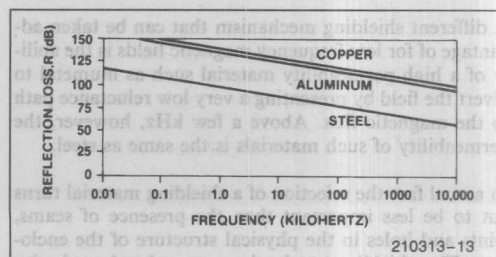


Figure 9. E-Field Shielding

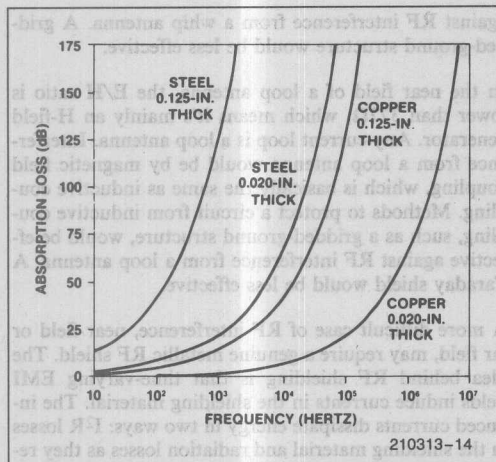


Figure 10. H-Field Shielding

Copper and aluminum both have the same permeability, but copper is slightly more conductive, and so provides slightly greater reflection loss to an E-field. Steel is less effective for two reasons. First, it has a somewhat elevated permeability due to its iron content, and second, as tends to be the case with magnetic materials, it is less conductive.

On the other hand, according to the expression for absorption loss to an H-field, H-field shielding is more effective at higher frequencies and with shield material that has both high conductivity and high permeability. In practice, however, selecting steel for its high permeability involves some compromise in conductivity. But the increase in permeability more than makes up for the decrease in conductivity, as can be seen in Figure 10. This figure also shows the effect of shield thickness.

A composite of E-field and H-field shielding is shown in Figure 11. However, this type of data is meaningful only in the far field. In the near field the EMI could be 90% H-field, in which case the reflection loss is irrelevant. It would be advisable then to beef up the absorption loss, at the expense of reflection loss, by choosing steel. A better conductor than steel might be less expensive, but quite ineffective.

A different shielding mechanism that can be taken advantage of for low frequency magnetic fields is the ability of a high permeability material such as mumetal to divert the field by presenting a very low reluctance path to the magnetic flux. Above a few kHz, however, the permeability of such materials is the same as steel.

In actual fact the selection of a shielding material turns out to be less important than the presence of seams, joints and holes in the physical structure of the enclosure. The shielding mechanisms are related to the induction of currents in the shield material, but the cur-

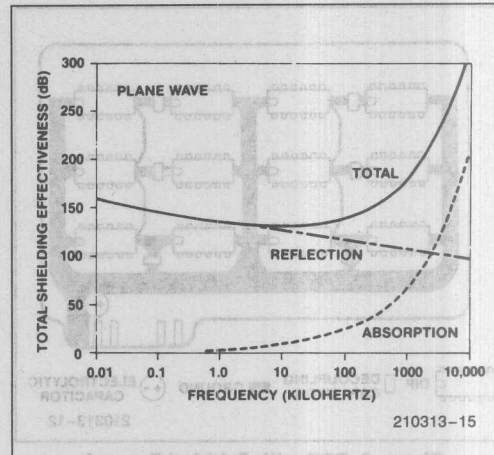


Figure 11. E- and H-Field Shielding

rents must be allowed to flow freely. If they have to detour around slots and holes, as shown in Figure 12, the shield loses much of its effectiveness.

As can be seen in Figure 12, the severity of the detour has less to do with the area of the hole than it does with the geometry of the hole. Comparing Figure 12c with 12d shows that a long narrow discontinuity such as a seam can cause more RF leakage than a line of holes with larger total area. A person who is responsible for designing or selecting rack or chassis enclosures for an EMI environment needs to be familiar with the techniques that are available for maintaining electrical continuity across seams. Information on these techniques is available in the references.

Grounds

There are two kinds of grounds: earth-ground and signal ground. The earth is not an equipotential surface, so earth ground potential varies. That and its other electrical properties are not conducive to its use as a return conductor in a circuit. However, circuits are often connected to earth ground for protection against shock hazards. The other kind of ground, signal ground, is an arbitrarily selected reference node in a circuit—the node with respect to which other node voltages in the circuit are measured.

SAFETY GROUND

The standard 3-wire single-phase AC power distribution system is represented in Figure 13. The white wire is earth-grounded at the service entrance. If a load circuit has a metal enclosure or chassis, and if the black wire develops a short to the enclosure, there will be a shock hazard to operating personnel, unless the enclosure itself is earth-grounded. If the enclosure is earth-

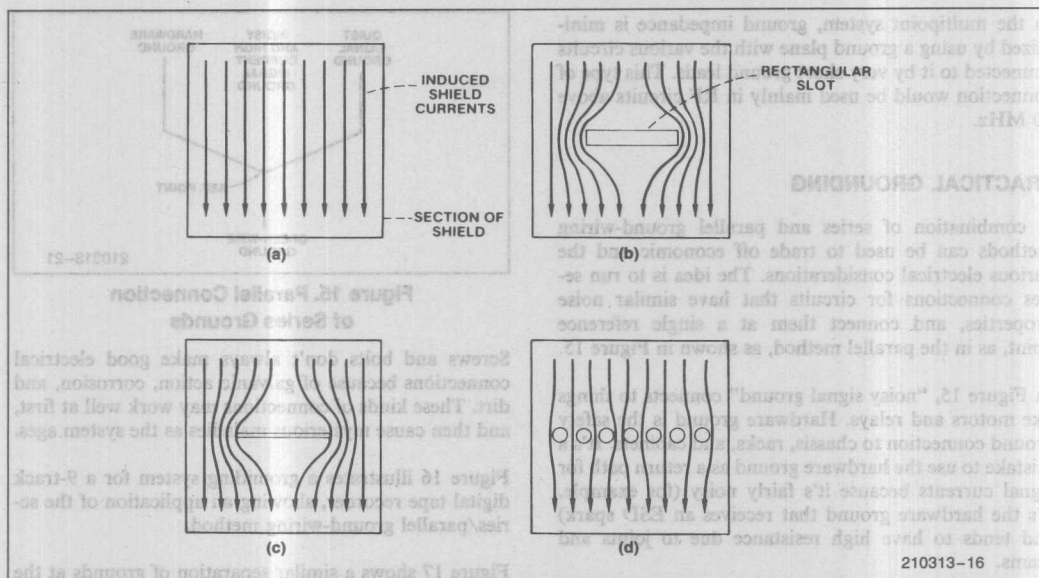


Figure 12. Effect of Shield Discontinuity on Magnetically Induced Shield Current

grounded, a short results in a blown fuse rather than a "hot" enclosure. The earth-ground connection to the enclosure is called a safety ground. The advantage of the 3-wire power system is that it distributes a safety ground along with the power.

Note that the safety-ground wire carries no current, except in case of a fault, so that at least for low frequencies it's at earth-ground potential along its entire length. The white wire, on the other hand, may be several volts off ground, due to the IR drop along its length.

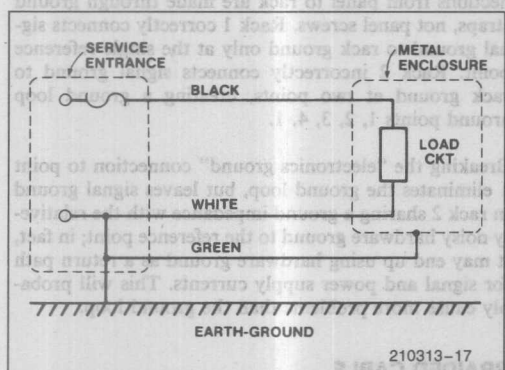


Figure 13. Single-Phase Power Distribution

SIGNAL GROUND

Signal ground is a single point in a circuit that is designated to be the reference node for the circuit. Commonly, wires that connect to this single point are also referred to as "signal ground." In some circles "power supply common" or PSC is the preferred terminology for these conductors. In any case, the manner in which these wires connect to the actual reference point is the basis of distinction among three kinds of signal-ground wiring methods: series, parallel, and multipoint. These methods are shown in Figure 14.

The series connection is pretty common because it's simple and economical. It's the noisiest of the three, however, due to common ground impedance coupling between the circuits. When several circuits share a ground wire, currents from one circuit, flowing through the finite impedance of the common ground line, cause variations in the ground potential of the other circuits. Given that the currents in a digital system tend to be spiked, and that the common impedance is mainly inductive reactance, the variations could be bad enough to cause bit errors in high current or particularly noisy situations.

The parallel connection eliminates common ground impedance problems, but uses a lot of wire. Other disadvantages are that the impedance of the individual ground-lines can be very high, and the ground lines themselves can become sources of EMI.

In the multipoint system, ground impedance is minimized by using a ground plane with the various circuits connected to it by very short ground leads. This type of connection would be used mainly in RF circuits above 10 MHz.

PRACTICAL GROUNDING

A combination of series and parallel ground-wiring methods can be used to trade off economic and the various electrical considerations. The idea is to run series connections for circuits that have similar noise properties, and connect them at a single reference point, as in the parallel method, as shown in Figure 15.

In Figure 15, "noisy signal ground" connects to things like motors and relays. Hardware ground is the safety ground connection to chassis, racks, and cabinets. It's a mistake to use the hardware ground as a return path for signal currents because it's fairly noisy (for example, it's the hardware ground that receives an ESD spark) and tends to have high resistance due to joints and seams.

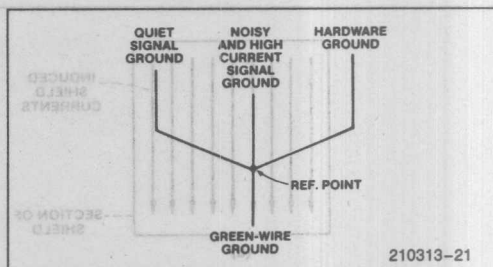


Figure 15. Parallel Connection of Series Grounds

Screws and bolts don't always make good electrical connections because of galvanic action, corrosion, and dirt. These kinds of connections may work well at first, and then cause mysterious maladies as the system ages.

Figure 16 illustrates a grounding system for a 9-track digital tape recorder, showing an application of the series/parallel ground-wiring method.

Figure 17 shows a similar separation of grounds at the PCB level. Currents in multiplexed LED displays tend to put a lot of noise on the ground and supply lines because of the constant switching and changing involved in the scanning process. The segment driver ground is relatively quiet, since it doesn't conduct the LED currents. The digit driver ground is noisier, and should be provided with a separate path to the PCB ground terminal, even if the PCB ground layout is gridded. The LED feed and return current paths should be laid out on opposite sides of the board like parallel flat conductors.

Figure 18 shows right and wrong ways to make ground connections in racks. Note that the safety ground connections from panel to rack are made through ground straps, not panel screws. Rack 1 correctly connects signal ground to rack ground only at the single reference point. Rack 2 incorrectly connects signal ground to rack ground at two points, creating a ground loop around points 1, 2, 3, 4, 1.

Breaking the "electronics ground" connection to point 1 eliminates the ground loop, but leaves signal ground in rack 2 sharing a ground impedance with the relatively noisy hardware ground to the reference point; in fact, it may end up using hardware ground as a return path for signal and power supply currents. This will probably cause more problems than the ground loop.

BRAIDED CABLE

Ground impedance problems can be virtually eliminated by using braided cable. The reduction in impedance is due to skin effect: At higher frequencies the current tends to flow along the surface of a conductor rather

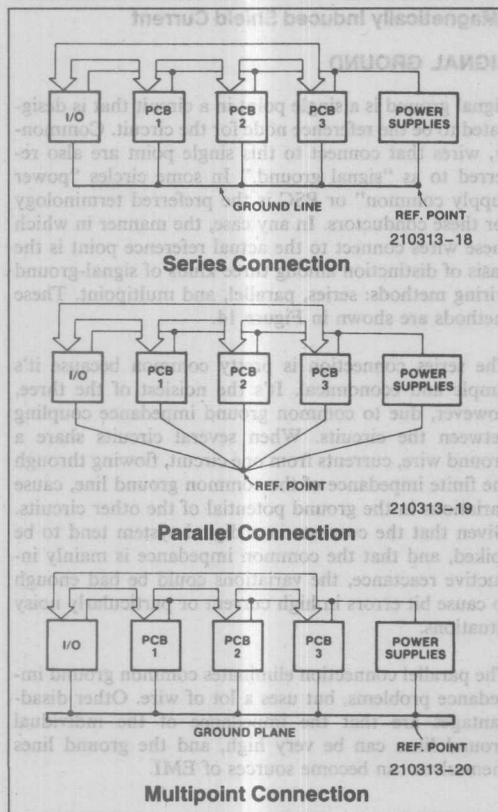


Figure 14. Three Ways to Wire the Grounds

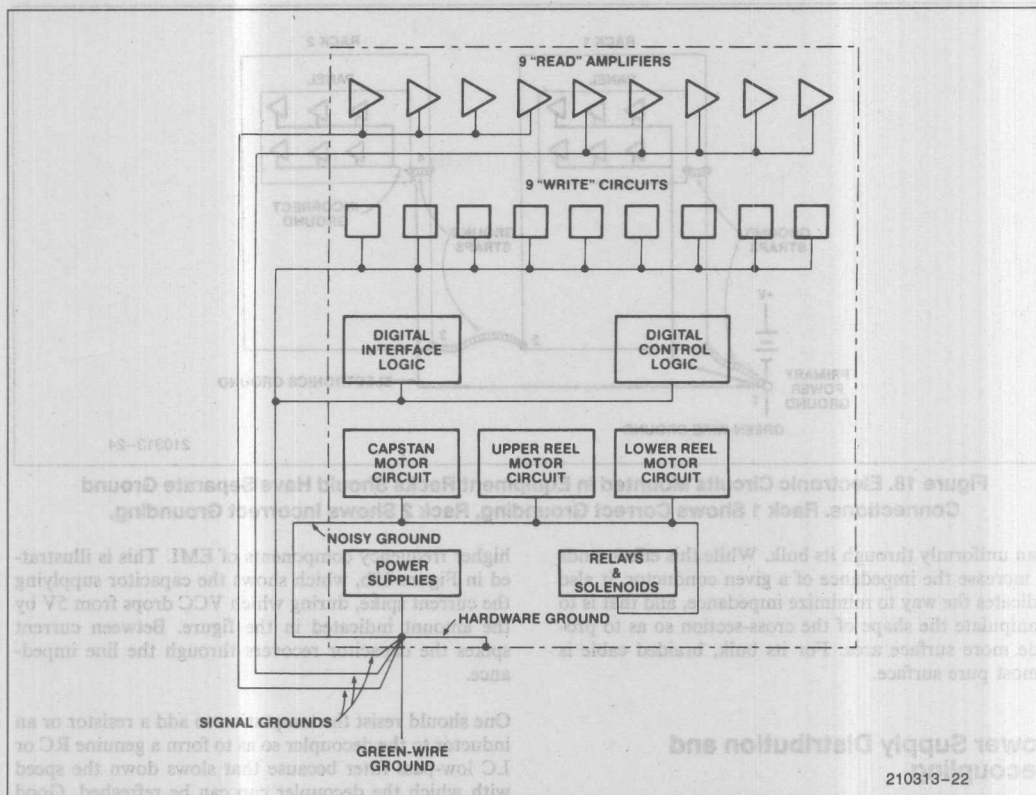


Figure 16. Ground System in a 9-Track Digital Recorder

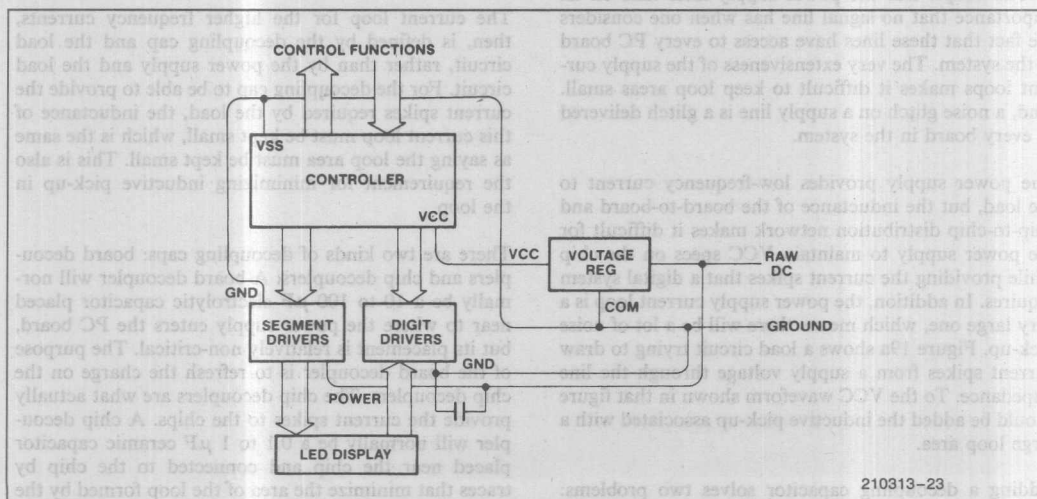


Figure 17. Separate Ground for Multiplexed LED Display

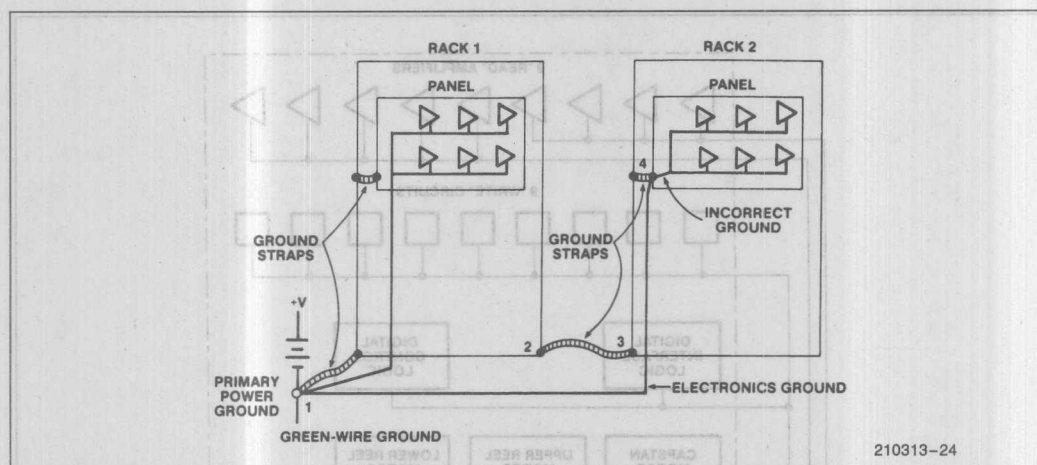


Figure 18. Electronic Circuits Mounted in Equipment Racks Should Have Separate Ground Connections. Rack 1 Shows Correct Grounding, Rack 2 Shows Incorrect Grounding.

than uniformly through its bulk. While this effect tends to increase the impedance of a given conductor, it also indicates the way to minimize impedance, and that is to manipulate the shape of the cross-section so as to provide more surface area. For its bulk, braided cable is almost pure surface.

Power Supply Distribution and Decoupling

The main consideration for power supply distribution lines is, as for signal lines, to minimize the areas of the current loops. But the power supply lines take on an importance that no signal line has when one considers the fact that these lines have access to every PC board in the system. The very extensiveness of the supply current loops makes it difficult to keep loop areas small. And, a noise glitch on a supply line is a glitch delivered to every board in the system.

The power supply provides low-frequency current to the load, but the inductance of the board-to-board and chip-to-chip distribution network makes it difficult for the power supply to maintain VCC specs on the chip while providing the current spikes that a digital system requires. In addition, the power supply current loop is a very large one, which means there will be a lot of noise pick-up. Figure 19a shows a load circuit trying to draw current spikes from a supply voltage through the line impedance. To the VCC waveform shown in that figure should be added the inductive pick-up associated with a large loop area.

Adding a decoupling capacitor solves two problems: The capacitor acts as a nearby source of charge to supply the current spikes through a smaller line impedance, and it defines a much smaller loop area for the

higher frequency components of EMI. This is illustrated in Figure 19b, which shows the capacitor supplying the current spike, during which VCC drops from 5V by the amount indicated in the figure. Between current spikes the capacitor recovers through the line impedance.

One should resist the temptation to add a resistor or an inductor to the decoupler so as to form a genuine RC or LC low-pass filter because that slows down the speed with which the decoupler cap can be refreshed. Good filtering and good decoupling are not necessarily the same thing.

The current loop for the higher frequency currents, then, is defined by the decoupling cap and the load circuit, rather than by the power supply and the load circuit. For the decoupling cap to be able to provide the current spikes required by the load, the inductance of this current loop must be kept small, which is the same as saying the loop area must be kept small. This is also the requirement for minimizing inductive pick-up in the loop.

There are two kinds of decoupling caps: board decouplers and chip decouplers. A board decoupler will normally be a 10 to 100 μ F electrolytic capacitor placed near to where the power supply enters the PC board, but its placement is relatively non-critical. The purpose of the board decoupler is to refresh the charge on the chip decouplers. The chip decouplers are what actually provide the current spikes to the chips. A chip decoupler will normally be a 0.1 to 1 μ F ceramic capacitor placed near the chip and connected to the chip by traces that minimize the area of the loop formed by the cap and the chip. If a chip decoupler is not properly placed on the board, it will be ineffective as a decoupler

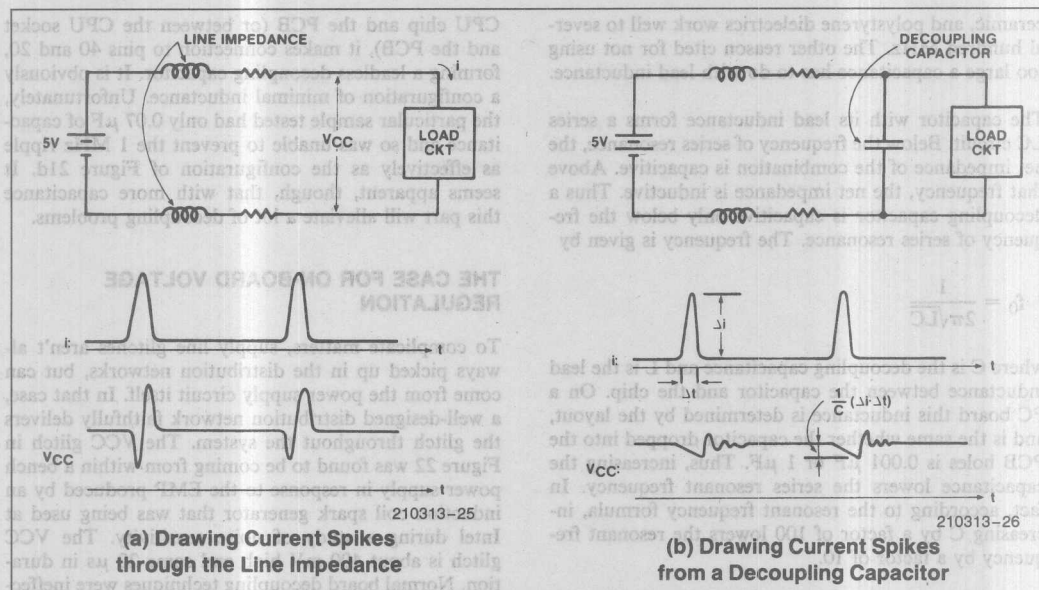


Figure 19. What a Decoupling Capacitor Does

and will serve only to increase the cost of the board. Good and bad placement of decoupling capacitors are illustrated in Figure 20.

Power distribution traces on the PC board need to be laid out so as to obtain minimal area (minimal inductance) in the loops formed by each chip and its decoupler, and by the chip decouplers and the board decoupler. One way to accomplish this goal is to use a power plane. A power plane is the same as a ground plane, but at VCC potential. More economically, a power grid similar to the ground grid previously discussed (Figure 8) can be used. Actually, if the chip decoupling loops are small, other aspects of the power layout are less critical. In other words, power planes and power gridding aren't needed, but power traces *should* be laid in the closest possible proximity to ground traces, prefer-

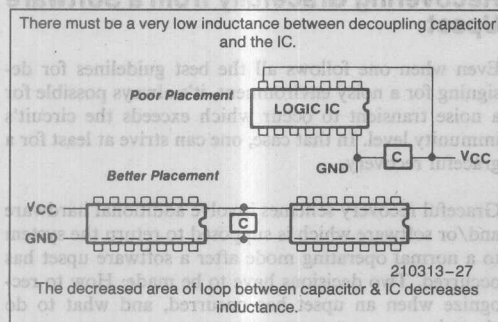


Figure 20. Placement of Decoupling Capacitors

ably so that each power trace is on the direct opposite side of the board from a ground trace.

Special-purpose power supply distribution buses which mount on the PCB are available. The buses use a parallel flat conductor configuration, one conductor being a VCC line and the other a ground line. Used in conjunction with a gridded ground layout, they not only provide a low-inductance distribution system, but can themselves form part of the ground grid, thus facilitating the PCB layout. The buses are available with and without enhanced bus capacitance, under the names Mini/Bus® and Q/PAC® from Rogers Corp. (5750 E. McKellips, Mesa, AZ 85205).

SELECTING THE VALUE OF THE DECOUPLING CAP

The effectiveness of the decoupling cap has a lot to do with the way the power and ground traces connect this capacitor to the chip. In fact, the area formed by this loop is more important than the value of the capacitance. Then, given that the area of this loop is indeed minimal, it can generally be said that the larger the value of the decoupling cap, the more effective it is, if the cap has a mica, ceramic, glass, or polystyrene dielectric.

It's often said, and not altogether accurately, that the chip decoupler shouldn't have too large a value. There are two reasons for this statement. One is that some capacitors, because of the nature of their dielectrics, tend to become inductive or lossy at higher frequencies. This is true of electrolytic capacitors, but mica, glass,

al hundred MHz. The other reason cited for not using too large a capacitance has to do with lead inductance.

The capacitor with its lead inductance forms a series LC circuit. Below the frequency of series resonance, the net impedance of the combination is capacitive. Above that frequency, the net impedance is inductive. Thus a decoupling capacitor is capacitive only below the frequency of series resonance. The frequency is given by

$$f_0 = \frac{1}{2\pi\sqrt{LC}}$$

where C is the decoupling capacitance and L is the lead inductance between the capacitor and the chip. On a PC board this inductance is determined by the layout, and is the same whether the capacitor dropped into the PCB holes is 0.001 μ F or 1 μ F. Thus, increasing the capacitance lowers the series resonant frequency. In fact, according to the resonant frequency formula, increasing C by a factor of 100 lowers the resonant frequency by a factor of 10.

Figures quoted on the series resonant frequency of a 0.01 μ F capacitor run from 10 to 15 MHz, depending on the lead length. If these numbers were accurate, a 1 μ F capacitor in the same position on the board would have a resonant frequency of 1.0 to 1.5 MHz, and as a decoupler would do more harm than good. However, the numbers are based on a presumed inductance of a given length of wire (the lead length). It should be noted that a "length of wire" has no inductance at all, strictly speaking. Only a complete current loop has inductance, and the inductance depends on the geometry of the loop. Figures quoted on the inductance of a length of wire are based on a presumably "very large" loop area, such that the magnetic field produced by the return current has no cancellation effect on the field produced by the current in the given length of wire. Such a loop geometry is not and should not be the case with the decoupling loop.

Figure 21 shows VCC waveforms, measured between pins 40 and 20 (VCC and VSS) of an 8751 CPU, for several conditions of decoupling on a PC board that has a decoupling loop area slightly larger than necessary. These photographs show the effects of increasing the decoupling capacitance and decreasing the area of the decoupling loop. The indications are that a 1 μ F capacitor is better than a 0.1 μ F capacitor, which in turn is better than nothing, and that the board should have been laid out with more attention paid to the area of the decoupling loop.

Figure 21e was obtained using a special-purpose experimental capacitor designed by Rogers Corp. (Q-Pac Division, Mesa, AZ) for use as a decoupler. It consists of two parallel plates, the length of a 40-pin DIP, separated by a ceramic dielectric. Sandwiched between the

and the PCB), it makes connection to pins 40 and 20, forming a leadless decoupling capacitor. It is obviously a configuration of minimal inductance. Unfortunately, the particular sample tested had only 0.07 μ F of capacitance and so was unable to prevent the 1 MHz ripple as effectively as the configuration of Figure 21d. It seems apparent, though, that with more capacitance this part will alleviate a lot of decoupling problems.

THE CASE FOR ON-BOARD VOLTAGE REGULATION

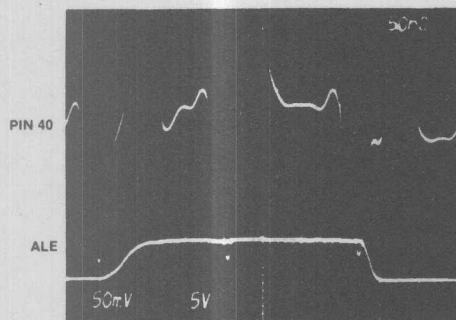
To complicate matters, supply line glitches aren't always picked up in the distribution networks, but can come from the power supply circuit itself. In that case, a well-designed distribution network faithfully delivers the glitch throughout the system. The VCC glitch in Figure 22 was found to be coming from within a bench power supply in response to the EMP produced by an induction coil spark generator that was being used at Intel during a study of noise sensitivity. The VCC glitch is about 400 mV high and some 20 μ s in duration. Normal board decoupling techniques were ineffective in removing it, but adding an on-board voltage regulator chip did the job.

Thus, a good case can be made in favor of using a voltage regulator chip on each PCB, instead of doing all the voltage regulation at the supply circuit. This eases requirements on the heat-sinking at the supply circuit, and alleviates much of the distribution and board decoupling headaches. However, it also brings in the possibility that different boards would be operating at slightly different VCC levels due to tolerance in the regulator chips; this then leads to slightly different logic levels from board to board. The implications of that may vary from nothing to latch-up, depending on what kinds of chips are on the boards, and how they react to an input "high" that is perhaps 0.4V higher than local VCC.

Recovering Gracefully from a Software Upset

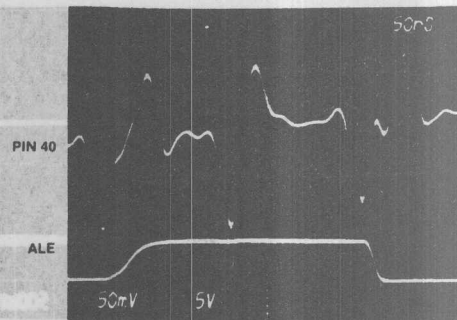
Even when one follows all the best guidelines for designing for a noisy environment, it's always possible for a noise transient to occur which exceeds the circuit's immunity level. In that case, one can strive at least for a graceful recovery.

Graceful recovery schemes involve additional hardware and/or software which is supposed to return the system to a normal operating mode after a software upset has occurred. Two decisions have to be made: How to recognize when an upset has occurred, and what to do about it.



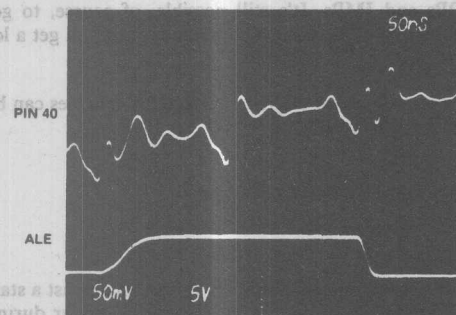
210313-28

(a) No Decoupling Cap



210313-29

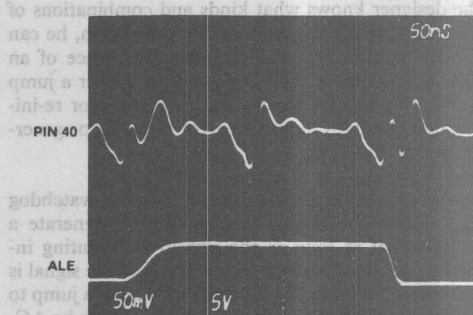
(b) 0.1 μ F Decoupler in Place on the PCB



210313-30

(c) 0.1 μ F Decoupler Stretched Directly from Pin 40 to Pin 20, under the Socket.

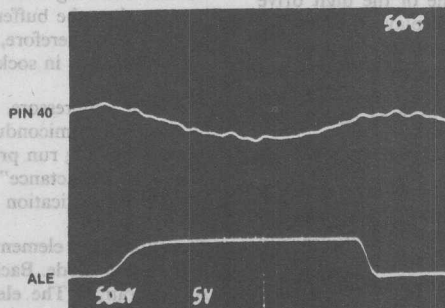
(The difference between this and 21b is due only to the change in loop geometry. Also shown is the upward slope of a ripple in VCC. The ripple frequency is 1 MHz, the same as ALE.)



210313-31

(d) 1.0 μ F Decoupler Stretched Directly from Pin 40 to Pin 20, under the Socket.

(This prevents the 1 MHz ripple, but there's no reduction in higher frequency components. Further increases in capacitance effected no further improvement.)



210313-32

(e) Special-Purpose Decoupling Cap under Development by Rogers Corp. (Further discussion in text.)

Figure 21. Noise on VCC Line

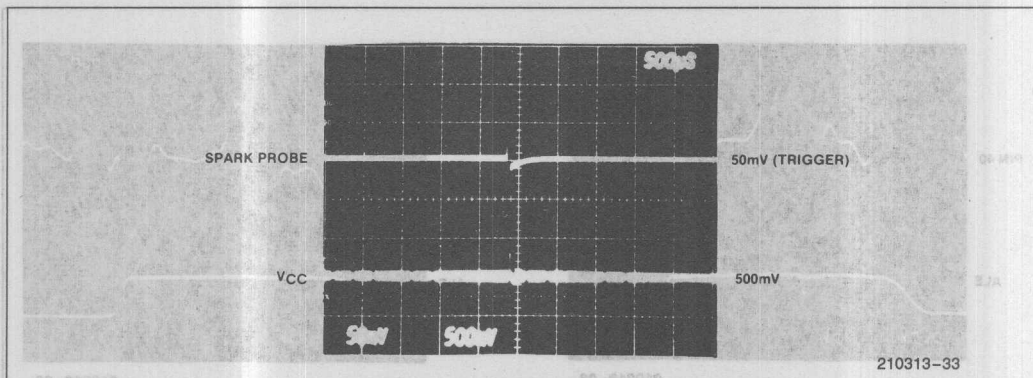


Figure 22. EMP-Induced Glitch

If the designer knows what kinds and combinations of outputs can legally be generated by the system, he can use gates to recognize and flag the occurrence of an illegal state of affairs. The flag can then trigger a jump to a recovery routine which then may check or re-initialize data, perhaps output an error message, or generate a simple reset.

The most reliable scheme is to use a so-called watchdog circuit. Here the CPU is programmed to generate a periodic signal as long as the system is executing instructions in an expected manner. The periodic signal is then used to hold off a circuit that will trigger a jump to a recovery routine. The periodic signal needs to be AC-coupled to the trigger circuit so that a "stuck-at" fault won't continue to hold off the trigger. Then, if the processor locks up someplace, the periodic signal is lost and the watchdog triggers a reset.

In practice, it may be convenient to drive the watchdog circuit with a signal which is being generated anyway by the system. One needs to be careful, however, that an upset does in fact discontinue that signal. Specifically, for example, one could use one of the digit drive signals going to a multiplexed display. But display scanning is often handled in response to a timer-interrupt, which may continue operating even though the main program is in a failure mode. Even so, with a little extra software, the signal can be used to control the watchdog (see Reference 8 on this).

Simpler schemes can work well for simpler systems. For example, if a CPU isn't doing anything but scanning and decoding a keyboard, there's little to lose and much to gain by simply resetting it periodically with an astable multivibrator. It only takes about 13 μ s (at 6 MHz) to reset an 8048 if the clock oscillator is already running.

A zero-cost measure is simply to fill all unused program memory with NOPs and Jumps to a recovery routine. The effectiveness of this method is increased by writing the program in segments that are separated by

NOPs and Jumps. It's still possible, of course, to get hung up in a data table or something. But you get a lot of protection, for the cost.

Further discussion of graceful recovery schemes can be found in Reference 13.

Special Problem Areas

ESD

MOS chips have some built-in protection against a static charge build-up on the pins, as would occur during normal handling, but there's no protection against the kinds of current levels and rise times that occur in a genuine electrostatic spark. These kinds of discharges can blow a crater in the silicon.

It must be recognized that connecting CPU pins unprotected to a keyboard or to anything else that is subject to electrostatic discharges makes an extremely fragile configuration. Buffering them is the very least one can do. But buffering doesn't completely solve the problem, because then the buffer chips will sustain the damage (even TTL); therefore, one might consider mounting the buffer chips in sockets for ease of replacement.

Transient suppressors, such as the TranZorbs® made by General Semiconductor Industries (Tempe, AZ), may in the long run provide the cheapest protection if their "zero inductance" structure is used. The structure and circuit application are shown in Figure 23.

The suppressor element is a pn junction that operates like a Zener diode. Back-to-back units are available for AC operation. The element is more or less an open circuit at normal system voltage (the standoff voltage rating for the device), and conducts like a Zener diode at the clamping voltage.

The lead inductance in the conventional transient suppressor package makes the conventional package essen-

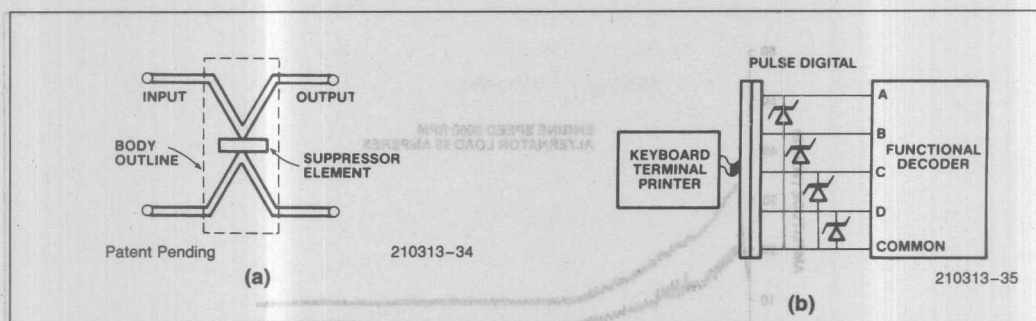


Figure 23. "Zero-Inductance" Structure and Use in Circuit

tially useless for protection against ESD pulses, owing to the fast rise of these pulses. The "zero inductance" units are available singly in a 4-pin DIP, and in arrays of four to a 16-pin DIP for PCB level protection. In that application they should be mounted in close proximity to the chips they protect.

In addition, metal enclosures or frames or parts that can receive an ESD spark should be connected by braided cable to the green-wire ground. Because of the ground impedance, ESD current shouldn't be allowed to flow through any signal ground, even if the chips are protected by transient suppressors. A 35 kV ESD spark can always spare a few hundred volts to drive a fast current pulse down a signal ground line if it can't find a braided cable to follow. Think how delighted your 8048 will be to find its VSS pin 250V higher than VCC for a few 10s of nanoseconds.

THE AUTOMOTIVE ENVIRONMENT

The automobile presents an extremely hostile environment for electronic systems. There are several parts to it:

1. Temperature extremes from -40°C to $+125^{\circ}\text{C}$ (under the hood) or $+85^{\circ}\text{C}$ (in the passenger compartment)
2. Electromagnetic pulses from the ignition system
3. Supply line transients that will knock your socks off

One needs to take a long, careful look at the temperature extremes. The allowable storage temperature range for most Intel MOS chips is -65°C to $+150^{\circ}\text{C}$, although some chips have a maximum storage temperature rating of $+125^{\circ}\text{C}$. In operation (or "under bias," as the data sheets say) the allowable ambient temperature range depends on the product grade, as follows:

Grade	Ambient Temperature	
	Min	Max
Commercial	0	70
Industrial	-40	$+85$
Automotive	-40	$+110$
Military	-55	$+125$

The different product grades are actually the same chip, but tested according to different standards. Thus, a given commercial-grade chip might actually pass military temperature requirements, but not have been tested for it. (Of course, there are other differences in grading requirements having to do with packaging, burn-in, traceability, etc.)

In any case, it's apparent that commercial-grade chips can't be used safely in automotive applications, not even in the passenger compartment. Industrial-grade chips can be used in the passenger compartment, and automotive or military chips are required in under-the-hood applications.

Ignition noise, CB radios, and that sort of thing are probably the least of your worries. In a poorly designed system, or in one that has not been adequately tested for the automotive environment, this type of EMI might cause a few software upsets, but not destroy chips.

The major problem, and the one that seems to come as the biggest surprise to most people, is the line transients. Regrettably, the 12V battery is not actually the source of power when the car is running. The charging system is, and it's not very clean. The only time the battery is the real source of power is when the car is first being started, and in that condition the battery terminals may be delivering about 5V or 6V. As follows is a brief description of the major idiosyncracies of the "12V" automotive power line.

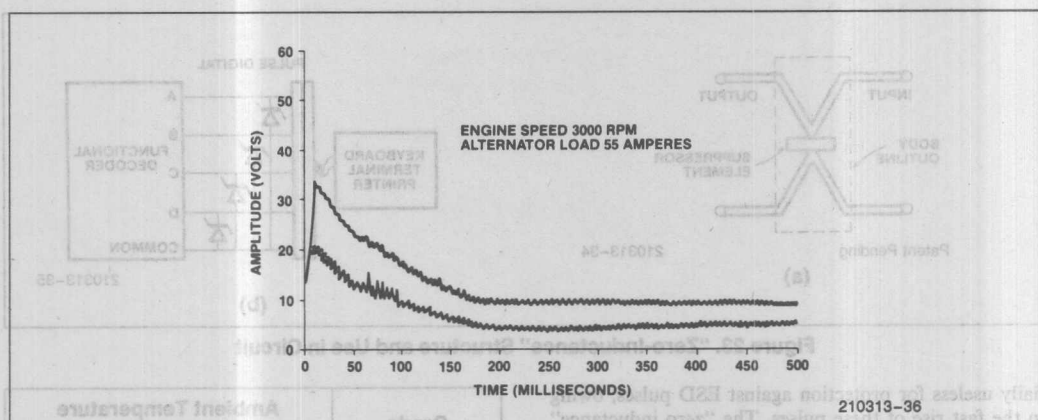


Figure 24. Typical Load Dump Transients

- An abrupt reduction in the alternator load causes a positive voltage transient called "load dump." In a load dump transient the line voltage rises to 20V or 30V in a few μ s, then decays exponentially with a time constant of about 100 μ s, as shown in Figure 24. Much higher peak voltages and longer decay times have also been reported. The worst case load dump is caused by disconnecting a low battery from the alternator circuit while the alternator is running. Normally this would happen intermittently when the battery terminal connections are defective.
- When the ignition is turned off, as the field excitation decays, the line voltage can go to between -40V and -100V for 100 μ s or more.
- Miscellaneous solenoid switching transients, such as the one shown in Figure 25, can drive the line to + or -200V to 400V for several μ s.

- Mutual coupling between unshielded wires in long harnesses can induce 100V and 200V transients in unprotected circuits.

What all this adds up to is that people in the business of building systems for automotive applications need a comprehensive testing program. An SAE guideline which describes the automotive environment is available to designers: SAE J1211, "Recommended Environmental Practices for Electronic Equipment Design," 1980 SAE Handbook, Part 1, pp. 22.80-22.96.

Some suggestions for protecting circuitry are shown in Figure 26. A transient suppressor is placed in front of the regulator chip to protect it. Since the rise times in these transients are not like those in ESD pulses, lead inductance is less critical and conventional devices can be used. The regulator itself is pretty much of a necessity, since a load dump transient is simply not going to be removed by any conventional LC or RC filter.

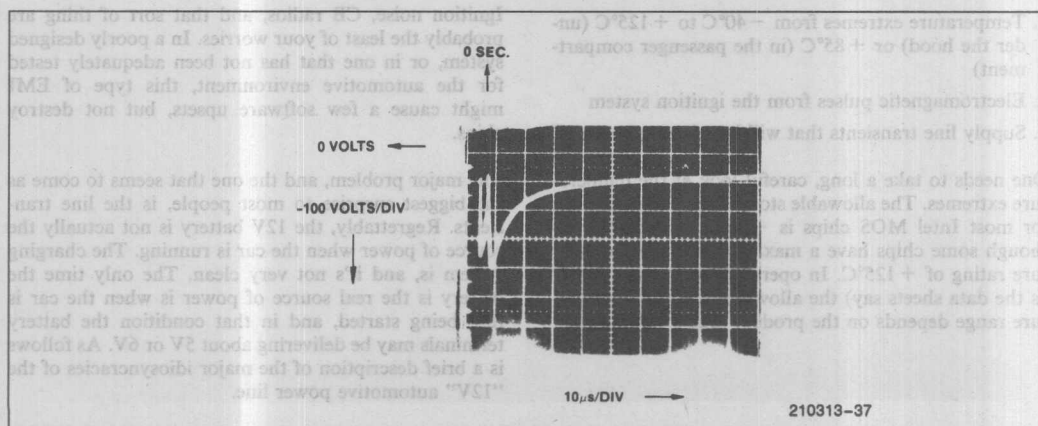


Figure 25. Transient Created by De-energizing an Air Conditioning Clutch Solenoid

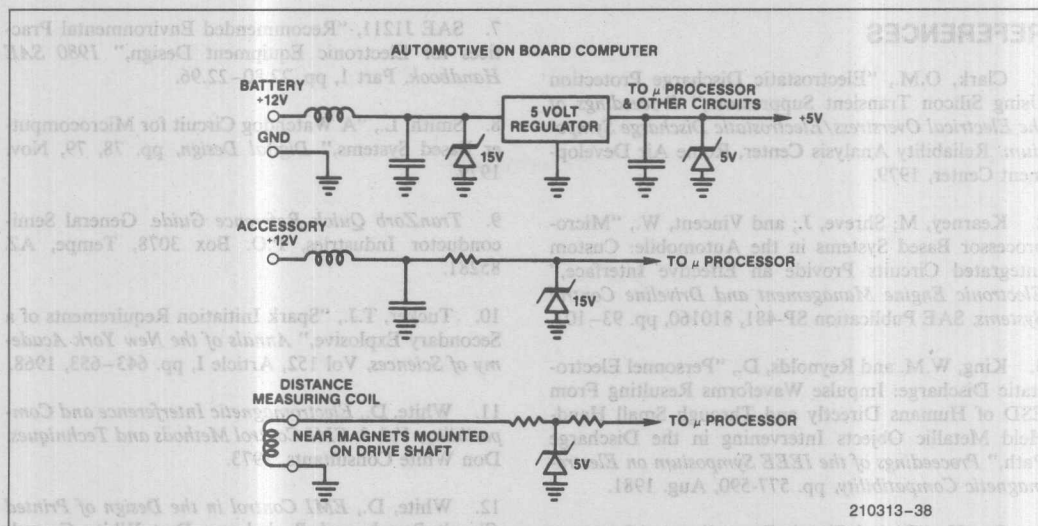


Figure 26. Use of Transient Suppressors in Automotive Applications

Special I/O interfacing is also required, because of the need for high tolerance to voltage transients, input noise, input/output isolation, etc. In addition, switches that are being monitored or driven by these buffers are usually referenced to chassis ground instead of signal ground, and in a car there can be many volts difference between the two. I/O interfacing is discussed in Reference 2.

The EMC Education committee has available a video tape: "Introduction to EMC—A Video Training Tape," by Henry Ott. Don White Consultants offers a series of training courses on many different aspects of electromagnetic compatibility. Most organizations that sponsor EMC courses also offer in-plant presentations.

Parting Thoughts

The main sources of information for this Application Note were the references by Ott and by White. Reference 5 is probably the finest treatment currently available on the subject. The other references provided specific information as cited in the text.

Courses and seminars on the subject of electromagnetic interference are given regularly throughout the year. Information on these can be obtained from:

IEEE Electromagnetic Compatibility Society
EMC Education Committee
345 East 47th Street
New York, NY 10017

Don White Consultants, Inc.
International Training Centre
P.O. Box D
Gainesville, VA 22065
Phone: (703) 347-0030

REFERENCES

1. Clark, O.M., "Electrostatic Discharge Protection Using Silicon Transient Suppressors," *Proceedings of the Electrical Overstress/Electrostatic Discharge Symposium*. Reliability Analysis Center, Rome Air Development Center, 1979.
2. Kearney, M; Shreve, J.; and Vincent, W., "Microprocessor Based Systems in the Automobile: Custom Integrated Circuits Provide an Effective Interface," *Electronic Engine Management and Driveline Control Systems*, SAE Publication SP-481, 810160, pp. 93-102.
3. King, W.M. and Reynolds, D., "Personnel Electrostatic Discharge: Impulse Waveforms Resulting From ESD of Humans Directly and Through Small Hand-Held Metallic Objects Intervening in the Discharge Path," *Proceedings of the IEEE Symposium on Electromagnetic Compatibility*, pp. 577-590, Aug. 1981.
4. Ott, H., "Digital Circuit Grounding and Interconnection," *Proceedings of the IEEE Symposium on Electromagnetic Compatibility*, pp. 292-297, Aug. 1981.
5. Ott, H., *Noise Reduction Techniques in Electronic Systems*. New York: Wiley, 1976.
6. 1981 *Interference Technology Engineers' Master (ITEM) Directory and Design Guide*. R. and B. Enterprises, P.O. Box 328, Plymouth Meeting, PA 19426.
7. SAE J1211, "Recommended Environmental Practices for Electronic Equipment Design," 1980 *SAE Handbook*, Part 1, pp. 22.80-22.96.
8. Smith, L., "A Watchdog Circuit for Microcomputer Based Systems," *Digital Design*, pp. 78, 79, Nov. 1979.
9. *TranZorb Quick Reference Guide*. General Semiconductor Industries, P.O. Box 3078, Tempe, AZ 85281.
10. Tucker, T.J., "Spark Initiation Requirements of a Secondary Explosive," *Annals of the New York Academy of Sciences*, Vol 152, Article I, pp. 643-653, 1968.
11. White, D., *Electromagnetic Interference and Compatibility, Vol. 3: EMI Control Methods and Techniques*. Don White Consultants, 1973.
12. White, D., *EMI Control in the Design of Printed Circuit Boards and Backplanes*. Don White Consultants, 1981.
13. Yarkoni, B. and Wharton, J., "Designing Reliable Software for Automotive Applications," *SAE Transactions*, 790237, July 1979.

Parting Thoughts

The main source of information for this Application Note were the references by Ott and by White. Reference 2 is probably the finest treatment currently available on the subject. The other references provided specific information as cited in the text.

Courses and seminars on the subject of electromagnetic interference are given regularly throughout the year. Information on these can be obtained from:

IEEE Electromagnetic Compatibility Society
EMC Education Committee
145 East 47th Street
New York, NY 10017

Don White Consultants, Inc.
International Training Centre
P.O. Box D
Gannville, VA 22063
Phone: (703) 347-0030

INTRODUCTION

Intel's microcontroller families (MCS®-48, MCS®-51, and iACX-96) contain a circuit that is commonly referred to as the "on-chip oscillator". The on-chip circuitry is not itself an oscillator, of course, but an amplifier that is suitable for use as the amplifier part of a feedback oscillator. The data sheets and Microcontroller Handbook show how the on-chip amplifier and several off-chip components can be used to design a working oscillator. With proper selection of off-chip components, these oscillator circuits will perform better than almost any other type of clock oscillator, and by almost any criterion of excellence. The suggested circuits are simple, economical, stable, and reliable.

We offer assistance to our customers in selecting suitable off-chip components to work with the on-chip oscillator circuitry. It should be noted, however, that Intel cannot assume the responsibility of writing specifications for the off-chip components of the complete oscillator circuit, nor of guaranteeing the performance of the finished design in production, anymore than a transistor manufacturer, whose data sheets show a number of suggested amplifier circuits, can assume responsibility for the operation, in production, of any of them.

We are often asked why we don't publish a list of required crystal or ceramic resonator specifications, and recommend values for the other off-chip components. This has been done in the past, but sometimes with consequences that were not intended.

Suppose we suggest a maximum crystal resistance of 30 ohms for some given frequency. Then your crystal supplier tells you the 30-ohm crystals are going to cost twice as much as 50-ohm crystals. Fearing that Intel will not "guarantee operation" with 50-ohm crystals, you order the expensive ones. In fact, Intel guarantees only what is embodied within an Intel product. Besides, there is no reason why 50-ohm crystals couldn't be used, if the other off-chip components are suitably adjusted.

Should we recommend values for the other off-chip components? Should we do it for 50-ohm crystals or 30-ohm crystals? With respect to what should we optimize their selection? Should we minimize start-up time or maximize frequency stability? In many applications, neither start-up time nor frequency stability are particularly critical, and our "recommendations" are only restricting your system to unnecessary tolerances. It all depends on the application.

Although we will neither "specify" nor "recommend" specific off-chip components, we do offer assistance in these tasks. Intel application engineers are available to provide whatever technical assistance may be needed or desired by our customers in designing with Intel products.

This Application Note is intended to provide such assistance in the design of oscillator circuits for microcontroller systems. Its purpose is to describe in a practical manner how oscillators work, how crystals and ceramic resonators work (and thus how to spec them), and what the on-chip amplifier looks like electronically and what its operating characteristics are. A BASIC program is provided in Appendix II to assist the designer in determining the effects of changing individual parameters. Suggestions are provided for establishing a pre-production test program.

FEEDBACK OSCILLATORS

Loop Gain

Figure 1 shows an amplifier whose output line goes into some passive network. If the input signal to the amplifier is v_1 , then the output signal from the amplifier is $v_2 = Av_1$ and the output signal from the passive network is $v_3 = \beta v_2 = \beta Av_1$. Thus βA is the overall gain from terminal 1 to terminal 3.

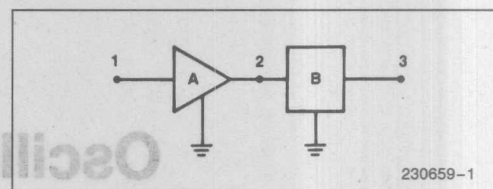


Figure 1. Factors in Loop Gain

Now connect terminal 1 to terminal 3, so that the signal path forms a loop: 1 to 2 to 3, which is also 1. Now we have a feedback loop, and the gain factor βA is called the *loop gain*.

Gain factors are complex numbers. That means they have a magnitude and a phase angle, both of which vary with frequency. When writing a complex number, one must specify both quantities, magnitude and angle. A number whose magnitude is 3, and whose angle is 45 degrees is commonly written this way: $3\angle 45^\circ$. The number 1 is, in complex number notation, $1\angle 0^\circ$, while -1 is $1\angle 180^\circ$.

By closing the feedback loop in Figure 1, we force the equality

$$v_1 = \beta Av_1$$

This equation has two solutions:

- 1) $v_1 = 0$;
- 2) $\beta A = 1\angle 0^\circ$.

In a given circuit, either or both of the solutions may be in effect. In the first solution the circuit is quiescent (no output signal). If you're trying to make an oscillator, a no-signal condition is unacceptable. There are ways to guarantee that the second solution is the one that will be in effect, and that the quiescent condition will be excluded.

How Feedback Oscillators Work

A feedback oscillator amplifies its own noise and feeds it back to itself in exactly the right phase, at the oscillation frequency, to build up and reinforce the desired oscillations. Its ability to do that depends on its loop gain. First, oscillations can occur only at the frequency for which the loop gain has a phase angle of 0 degrees. Second build-up of oscillations will occur only if the loop gain exceeds 1 at the frequency. Build-up continues until nonlinearities in the circuit reduce the average value of the loop gain to exactly 1.

Start-up characteristics depend on the small-signal properties of the circuit, specifically, the small-signal loop gain. Steady-state characteristics of the oscillator depend on the large-signal properties of the circuit, such as the transfer curve (output voltage vs. input voltage) of the amplifier, and the clamping effect of the input protection devices. These things will be discussed more fully further on. First we will look at the basic operation of the particular oscillator circuit, called the "positive reactance" oscillator.

The Positive Reactance Oscillator

Figure 2 shows the configuration of the positive reactance oscillator. The inverting amplifier, working into the impedance of the feedback network, produces an output signal that is nominally 180 degrees out of phase with its input. The feedback network must provide an additional 180 degrees phase shift, such that the overall loop gain has zero (or 360) degrees phase shift at the oscillation frequency.

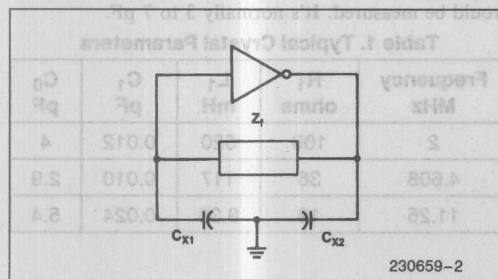


Figure 2. Positive Reactance Oscillator

In order for the loop gain to have zero phase angle it is necessary that the feedback element Z_f have a positive reactance. That is, it must be inductive. Then, the frequency at which the phase angle is zero is approximately the frequency at which

$$X_f = \frac{+1}{\omega C}$$

where X_f is the reactance of Z_f (the total Z_f being $R_f + jX_f$), and C is the series combination of C_{X1} and C_{X2} .

$$C = \frac{C_{X1} C_{X2}}{C_{X1} + C_{X2}}$$

In other words, Z_f and C form a parallel resonant circuit.

If Z_f is an inductor, then $X_f = \omega L$, and the frequency at which the loop gain has zero phase is the frequency at which

$$\omega L = \frac{1}{\omega C}$$

or

$$\omega = \frac{1}{\sqrt{LC}}$$

Normally, Z_f is not an inductor, but it must still have a positive reactance in order for the circuit to oscillate. There are some piezoelectric devices on the market that show a positive reactance, and provide a more stable oscillation frequency than an inductor will. Quartz crystals can be used where the oscillation frequency is critical, and lower cost ceramic resonators can be used where the frequency is less critical.

When the feedback element is a piezoelectric device, this circuit configuration is called a Pierce oscillator. The advantage of piezoelectric resonators lies in their property of providing a wide range of positive reactance values over a very narrow range of frequencies. The reactance will equal $1/\omega C$ at some frequency within this range, so the oscillation frequency will be within the same range. Typically, the width of this range is

Only 0.5% of the nominal frequency of a quartz crystal, and about 3% of the nominal frequency of a ceramic resonator. With relatively little design effort, frequency accuracies of 0.03% or better can be obtained with quartz crystals, and 0.3% or better with ceramic resonators.

QUARTZ CRYSTALS

The crystal resonator is a thin slice of quartz sandwiched between two electrodes. Electrically, the device looks pretty much like a 5 or 6 pF capacitor, except that over certain ranges of frequencies the crystal has a positive (i.e., inductive) reactance.

The ranges of positive reactance originate in the piezoelectric property of quartz: Squeezing the crystal generates an internal E-field. The effect is reversible: Applying an AC E-field causes the crystal to vibrate. At certain vibrational frequencies there is a mechanical resonance. As the E-field frequency approaches a frequency of mechanical resonance, the measured reactance of the crystal becomes positive, as shown in Figure 3.

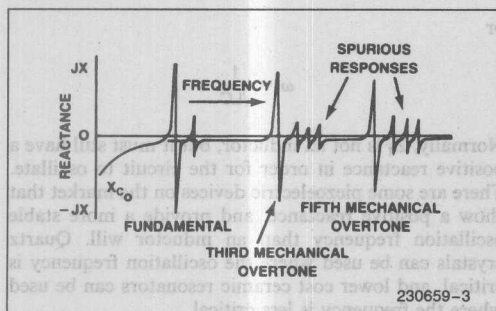


Figure 3. Crystal Reactance vs. Frequency

Typically there are several ranges of frequencies where in the reactance of the crystal is positive. Each range corresponds to a different mode of vibration in the crystal. The main resonances are the so-called fundamental response and the third and fifth overtone responses.

The overtone responses shouldn't be confused with the harmonics of the fundamental. They're not harmonics, but different vibrational modes. They're not in general at exact integer multiples of the fundamental frequency. There will also be "spurious" responses, occurring typically a few hundred KHz above each main response.

To assure that an oscillator starts in the desired mode on power-up, something must be done to suppress the loop gain in the undesired frequency ranges. The crystal itself provides some protection against unwanted modes of oscillation; too much resistance in that mode, for example. Additionally, junction capacitances in the amplifying devices tend to reduce the gain at higher frequencies, and thus may discriminate against unwanted modes. In some cases a circuit fix is necessary, such as inserting a trap, a phase shifter, or ferrite beads to kill oscillations in unwanted modes.

Crystal Parameters

Equivalent Circuit

Figure 4 shows an equivalent circuit that is used to represent the crystal for circuit analysis.

The R_1 - L_1 - C_1 branch is called the motivational arm of the crystal. The values of these parameters derive from the mechanical properties of the crystal and are constant for a given mode of vibration. Typical values for various nominal frequencies are shown in Table 1.

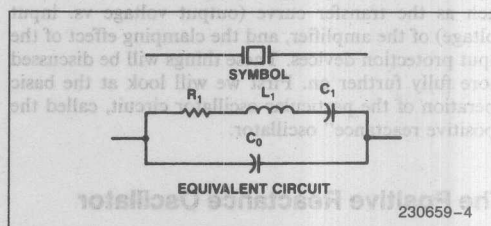


Figure 4. Quartz Crystal: Symbol and Equivalent Circuit

C_0 is called the shunt capacitance of the crystal. This is the capacitance of the crystal's electrodes and the mechanical holder. If one were to measure the reactance of the crystal at a frequency far removed from a resonance frequency, it is the reactance of this capacitance that would be measured. It's normally 3 to 7 pF.

Table 1. Typical Crystal Parameters

Frequency MHz	R_1 ohms	L_1 mH	C_1 pF	C_0 pF
2	100	520	0.012	4
4.608	36	117	0.010	2.9
11.25	19	8.38	0.024	5.4

The series resonant frequency of the crystal is the frequency at which L_1 and C_1 are in resonance. This frequency is given by

$$f_s = \frac{1}{2\pi\sqrt{L_1 C_1}}$$

At this frequency the impedance of the crystal is R_1 in parallel with the reactance of C_0 . For most purposes, this impedance is taken to be just R_1 , since the reactance of C_0 is so much larger than R_1 .

Load Capacitance

A crystal oscillator circuit such as the one shown in Figure 2 (redrawn in Figure 5) operates at the frequency for which the crystal is antiresonant (ie, parallel-resonant) with the total capacitance across the crystal terminals external to the crystal. This total capacitance external to the crystal is called the load capacitance.

As shown in Figure 5, the load capacitance is given by

$$C_L = \frac{C_{X1} C_{X2}}{C_{X1} + C_{X2}} + C_{\text{stray}}$$

The crystal manufacturer needs to know the value of C_L in order to adjust the crystal to the specified frequency.

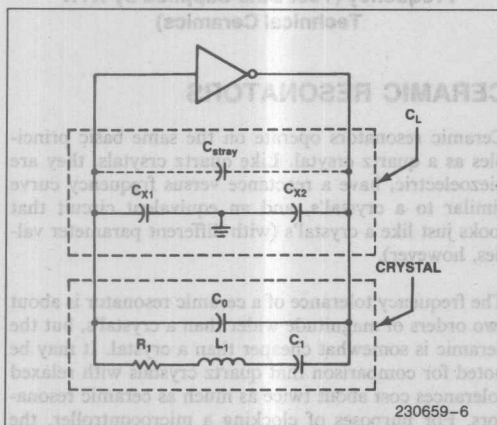


Figure 5. Load Capacitance

The adjustment involves putting the crystal in series with the specified C_L , and then "trimming" the crystal to obtain resonance of the series combination of the crystal and C_L at the specified frequency. Because of the high Q of the crystal, the resonant frequency of the series combination of the crystal and C_L is the same as

the antiresonant frequency of the parallel combination of the crystal and C_L . This frequency is given by

$$f_a = \frac{1}{2\pi\sqrt{L_1 C_1 (C_L + C_0) / (C_1 + C_L + C_0)}}$$

These frequency formulas are derived (in Appendix A) from the equivalent circuit of the crystal, using the assumptions that the Q of the crystal is extremely high, and that the circuit external to the crystal has no effect on the frequency other than to provide the load capacitance C_L . The latter assumption is not precisely true, but it is close enough for present purposes.

"Series" vs. "Parallel" Crystals

There is no such thing as a "series cut" crystal as opposed to a "parallel cut" crystal. There are different cuts of crystal, having to do with the parameters of its motional arm in various frequency ranges, but there is no special cut for series or parallel operation.

An oscillator is series resonant if the oscillation frequency is f_s of the crystal. To operate the crystal at f_s , the amplifier has to be noninverting. When buying a crystal for such an oscillator, one does not specify a load capacitance. Rather, one specifies the loading condition as "series."

If a "series" crystal is put into an oscillator that has an inverting amplifier, it will oscillate in parallel resonance with the load capacitance presented to the crystal by the oscillator circuit, at a frequency slightly above f_s . In fact, at approximately

$$f_a = f_s \left(1 + \frac{C_1}{2(C_L + C_0)} \right)$$

This frequency would typically be about 0.02% above f_s .

Equivalent Series Resistance

The "series resistance" often listed on quartz crystal data sheets is the real part of the crystal impedance at the crystal's calibration frequency. This will be R_1 if the calibration frequency is the series resonant frequency of the crystal. If the crystal is calibrated for parallel resonance with a load capacitance C_L , the equivalent series resistance will be

$$ESR = R_1 \left(1 + \frac{C_0}{C_L} \right)^2$$

The crystal manufacturer measures this resistance at the calibration frequency during the same operation in which the crystal is adjusted to the calibration frequency.

Frequency Tolerance

Frequency tolerance as discussed here is not a requirement on the crystal, but on the complete oscillator. There are two types of frequency tolerances on oscillators: frequency *accuracy* and frequency *stability*. Frequency accuracy refers to the oscillator's ability to run at an exact specified frequency. Frequency stability refers to the constancy of the oscillation frequency.

Frequency accuracy requires mainly that the oscillator circuit present to the crystal the same load capacitance that it was adjusted for. Frequency stability requires mainly that the load capacitance be constant.

In most digital applications the accuracy and stability requirements on the oscillator are so wide that it makes very little difference what load capacitance the crystal was adjusted to, or what load capacitance the circuit actually presents to the crystal. For example, if a crystal was calibrated to a load capacitance of 25 pF, and is used in a circuit whose actual load capacitance is 50 pF, the frequency error on that account would be less than 0.01%.

In a positive reactance oscillator, the crystal only needs to be in the intended response mode for the oscillator to satisfy a 0.5% or better frequency tolerance. That's because for any load capacitance the oscillation frequency is certain to be between the crystal's resonant and antiresonant frequencies.

Phase shifts that take place within the amplifier part of the oscillator will also affect frequency accuracy and stability. These phase shifts can normally be modeled as an "output capacitance" that, in the positive reactance oscillator, parallels C_{X2} . The predictability and constancy of this output capacitance over temperature and device sample will be the limiting factor in determining the tolerances that the circuit is capable of holding.

Drive Level

Drive level refers to the power dissipation in the crystal. There are two reasons for specifying it. One is that the parameters in the equivalent circuit are somewhat dependent on the drive level at which the crystal is calibrated. The other is that if the application circuit exceeds the test drive level by too much, the crystal may be damaged. Note that the terms "test drive level" and "rated drive level" both refer to the drive level at which the crystal is calibrated. Normally, in a microcontroller system, neither the frequency tolerances nor the power levels justify much concern for this specification. Some crystal manufacturers don't even require it for microprocessor crystals.

In a positive reactance oscillator, if one assumes the peak voltage across the crystal to be something in the neighborhood of V_{CC} , the power dissipation can be approximated as

$$P = 2R_1 [\pi f (C_L + C_0) V_{CC}]^2$$

This formula is derived in Appendix A. In a 5V system, P rarely evaluates to more than a milliwatt. Crystals with a standard 1 or 2 mW drive level rating can be used in most digital systems.

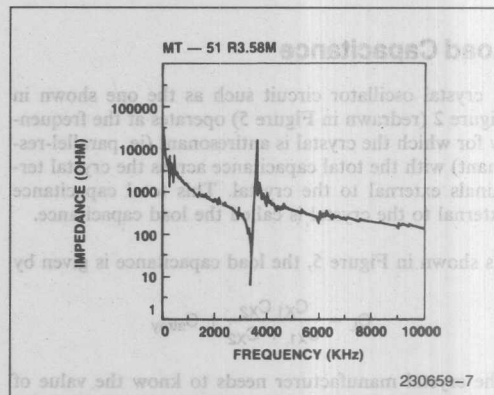


Figure 6. Ceramic Resonator Impedance vs. Frequency (Test Data Supplied by NTK Technical Ceramics)

CERAMIC RESONATORS

Ceramic resonators operate on the same basic principles as a quartz crystal. Like quartz crystals, they are piezoelectric, have a reactance versus frequency curve similar to a crystal's, and an equivalent circuit that looks just like a crystal's (with different parameter values, however).

The frequency tolerance of a ceramic resonator is about two orders of magnitude wider than a crystal's, but the ceramic is somewhat cheaper than a crystal. It may be noted for comparison that quartz crystals with relaxed tolerances cost about twice as much as ceramic resonators. For purposes of clocking a microcontroller, the frequency tolerance is often relatively noncritical, and the economic consideration becomes the dominant factor.

Figure 6 shows a graph of impedance magnitude versus frequency for a 3.58 MHz ceramic resonator. (Note that Figure 6 is a graph of $|Z_f|$ versus frequency, where

as Figure 3 is a graph of X_f versus frequency.) A number of spurious responses are apparent in Figure 6. The manufacturers state that spurious responses are more prevalent in the lower frequency resonators (kHz range) than in the higher frequency units (MHz range). For our purposes only the MHz range ceramics need to be considered.

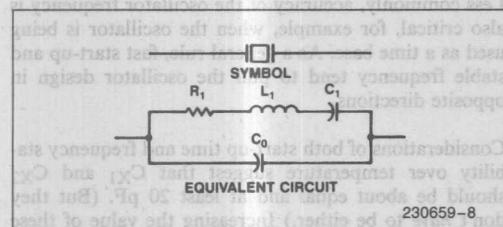


Figure 7. Ceramic Resonator: Symbol and Equivalent Circuit

Figure 7 shows the symbol and equivalent circuit for the ceramic resonator, both of which are the same as for the crystal. The parameters have different values, however, as listed in Table 2.

Table 2. Typical Ceramic Parameters

Frequency MHz	R_1 ohms	L_1 mH	C_1 pF	C_0 pF
3.58	7	0.113	19.6	140
6.0	8	0.094	8.3	60
8.0	7	0.092	4.6	40
11.0	10	0.057	3.9	30

Note that the motional arm of the ceramic resonator tends to have less resistance than the quartz crystal and also a vastly reduced L_1/C_1 ratio. This results in the motional arm having a Q (given by $(1/R_1) \sqrt{L_1/C_1}$) that is typically two orders of magnitude lower than that of a quartz crystal. The lower Q makes for a faster startup of the oscillator and for a less closely controlled frequency (meaning that circuitry external to the resonator will have more influence on the frequency than with a quartz crystal).

Another major difference is that the shunt capacitance of the ceramic resonator is an order of magnitude higher than C_0 of the quartz crystal and more dependent on the frequency of the resonator.

The implications of these differences are not all obvious, but some will be indicated in the section on Oscillator Calculations.

Specifications for Ceramic Resonators

Ceramic resonators are easier to specify than quartz crystals. All the vendor wants to know is the desired

frequency and the chip you want it to work with. They'll supply the resonators, a circuit diagram showing the positions and values of other external components that may be required and a guarantee that the circuit will work properly at the specified frequency.

OSCILLATOR DESIGN CONSIDERATIONS

Designers of microcontroller systems have a number of options to choose from for clocking the system. The main decision is whether to use the "on-chip" oscillator or an external oscillator. If the choice is to use the on-chip oscillator, what kinds of external components are needed to make it operate as advertised? If the choice is to use an external oscillator, what type of oscillator should it be?

The decisions have to be based on both economic and technical requirements. In this section we'll discuss some of the factors that should be considered.

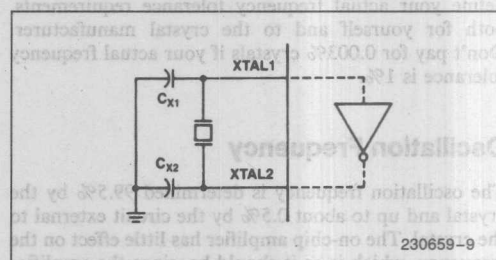


Figure 8. Using the "On-Chip" Oscillator

On-Chip Oscillators

In most cases, the on-chip amplifier with the appropriate external components provides the most economical solution to the clocking problem. Exceptions may arise in severe environments when frequency tolerances are tighter than about 0.01%.

The external components that need to be added are a positive reactance (normally a crystal or ceramic resonator) and the two capacitors C_{X1} and C_{X2} , as shown in Figure 8.

Crystal Specifications

Specifications for an appropriate crystal are not very critical, unless the frequency is. Any fundamental-mode crystal of medium or better quality can be used.

We are often asked what maximum crystal resistance should be specified. The best answer to this question is the lower the better, but use what's available. The crystal resistance will have some effect on start-up time and steady-state amplitude, but not so much that it can't be compensated for by appropriate selection of the capacitances C_{X1} and C_{X2} .

Similar questions are asked about specifications of load capacitance and shunt capacitance. The best advice we can give is to understand what these parameters mean and how they affect the operation of the circuit (that being the purpose of this Application Note), and then decide for yourself if such specifications are meaningful in your application or not. Normally, they're not, unless your frequency tolerances are tighter than about 0.1%.

Part of the problem is that crystal manufacturers are accustomed to talking "ppm" tolerances with radio engineers and simply won't take your order until you've filled out their list of specifications. It will help if you define your actual frequency tolerance requirements, both for yourself and to the crystal manufacturer. Don't pay for 0.003% crystals if your actual frequency tolerance is 1%.

Oscillation Frequency

The oscillation frequency is determined 99.5% by the crystal and up to about 0.5% by the circuit external to the crystal. The on-chip amplifier has little effect on the frequency, which is as it should be, since the amplifier parameters are temperature and process dependent.

The influence of the on-chip amplifier on the frequency is by means of its input and output (pin-to-ground) capacitances, which parallel C_{X1} and C_{X2} , and the XTAL1-to-XTAL2 (pin-to-pin) capacitance, which parallels the crystal. The input and pin-to-pin capacitances are about 7 pF each. Internal phase deviations from the nominal 180° can be modeled as an output capacitance of 25 to 30 pF. These deviations from the ideal have less effect in the positive reactance oscillator (with the inverting amplifier) than in a comparable series resonant oscillator (with the noninverting amplifier) for two reasons: first, the effect of the output capacitance is lessened, if not swamped, by the off-chip capacitor; secondly, the positive reactance oscillator is less sensitive, frequency-wise, to such phase errors.

Selection of C_{X1} and C_{X2}

Optimal values for the capacitors C_{X1} and C_{X2} depend on whether a quartz crystal or ceramic resona-

tor is being used, and also on application-specific requirements on start-up time and frequency tolerance.

Start-up time is sometimes more critical in microcontroller systems than frequency stability, because of various reset and initialization requirements.

Less commonly, accuracy of the oscillator frequency is also critical, for example, when the oscillator is being used as a time base. As a general rule, fast start-up and stable frequency tend to pull the oscillator design in opposite directions.

Considerations of both start-up time and frequency stability over temperature suggest that C_{X1} and C_{X2} should be about equal and at least 20 pF. (But they don't *have* to be either.) Increasing the value of these capacitances above some 40 or 50 pF improves frequency stability. It also tends to increase the start-up time. There is a maximum value (several hundred pF, depending on the value of R_1 of the quartz or ceramic resonator) above which the oscillator won't start up at all.

If the on-chip amplifier is a simple inverter, such as in the 8051, the user can select values for C_{X1} and C_{X2} between some 20 and 100 pF, depending on whether start-up time or frequency stability is the more critical parameter in a specific application. If the on-chip amplifier is a Schmitt Trigger, such as in the 8048, smaller values of C_{X1} must be used (5 to 30 pF), in order to prevent the oscillator from running in a relaxation mode.

Later sections in this Application Note will discuss the effects of varying C_{X1} and C_{X2} (as well as other parameters), and will have more to say on their selection.

Placement of Components

Noise glitches arriving at XTAL1 or XTAL2 pins at the wrong time can cause a miscount in the internal clock-generating circuitry. These kinds of glitches can be produced through capacitive coupling between the oscillator components and PCB traces carrying digital signals with fast rise and fall times. For this reason, the oscillator components should be mounted close to the chip and have short, direct traces to the XTAL1, XTAL2, and VSS pins.

Clocking Other Chips

There are times when it would be desirable to use the on-chip oscillator to clock other chips in the system.

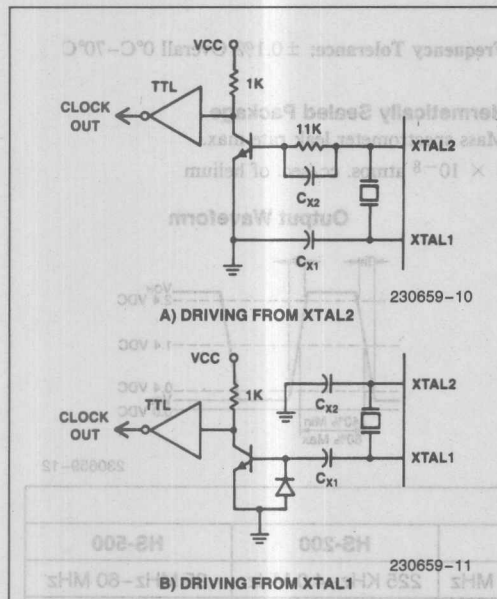


Figure 9. Using the On-Chip Oscillator to Drive Other Chips

This can be done if an appropriate buffer is used. A TTL buffer puts too much load on the on-chip amplifier for reliable start-up. A CMOS buffer (such as the 74HC04) can be used, if it's fast enough and if its V_{IH} and V_{IL} specs are compatible with the available signal amplitudes. Circuits such as shown in Figure 9 might also be considered for these types of applications.

Clock-related signals are available at the TO pin in the MCS-48 products, at ALE in the MCS-48 and MCS-51 lines, and the iACX-96 controllers provide a CLKOUT signal.

External Oscillators

When technical requirements dictate the use of an external oscillator, the external drive requirements for the microcontroller, as published in the data sheet, must be carefully noted. The logic levels are not in general TTL-compatible. And each controller has its idiosyncracies in this regard. The 8048, for example, requires that both XTAL1 and XTAL2 be driven. The 8051 can be driven that way, but the data sheet suggest the simpler method of grounding XTAL1 and driving XTAL2. For this method, the driving source must be capable of sinking some current when XTAL2 is being driven low.

For the external oscillator itself, there are basically two choices: ready-made and home-grown.

H2-100				Output Load
3.8 MHz-50 MHz	50 + MHz-30 MHz	255 KHz-4.0 MHz	25 MHz-60 MHz	
1.0 min V _A S + +0.4 V max ^a 80V40 ^a	1.0 min V _A S + +0.5 V max ^a 80V40 ^a	1.0 min V _A S + +0.4 V max ^a 80V40 ^a	1.0 min V _A S + +0.5 V max ^a 80V40 ^a	Output Current
18 mA min	18 mA min	18 mA min	18 mA min	Output Short
1 to 10 TTL Loads ^b	1 to 10 TTL Loads ^b	1 to 10 TTL Loads ^b	1 to 10 TTL Loads ^b	Fall Time
< 10 ns	< 10 ns	< 10 ns	< 10 ns	T _r (F _{rise} &
80V40 ^a	80V40 ^a	80V40 ^a	80V40 ^a	Symmetry
80V40 ^a	80V40 ^a	80V40 ^a	80V40 ^a	V _{OL} (Logic "0")
80V40 ^a	80V40 ^a	80V40 ^a	80V40 ^a	V _{OH} (Logic "1")
80V40 ^a	80V40 ^a	80V40 ^a	80V40 ^a	

Figure 10. Pre-Packaged Oscillator Data*

TTL Crystal Clock Oscillator

The HS-100, HS-200, & HS-500 all-metal package series of oscillators are TTL compatible & fit a DIP layout. Standard electrical specifications are shown below. Variations are available for special applications.

Frequency Range: HS-100—3.5 MHz to 30 MHz

HS-200—225 KHz to 3.5 MHz

HS-500—25 MHz to 60 MHz

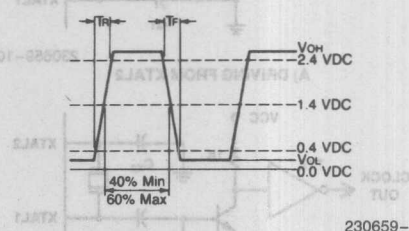
Frequency Tolerance: $\pm 0.1\%$ Overall $0^{\circ}\text{C}-70^{\circ}\text{C}$

Hermetically Sealed Package

Mass spectrometer leak rate max.

1×10^{-8} atmos. cc/sec. of helium

Output Waveform



230659-12

INPUT				
	HS-100		HS-200	HS-500
	3.5 MHz-20 MHz	20+ MHz-30 MHz	225 KHz-4.0 MHz	25 MHz-60 MHz
Supply Voltage (V _{CC})	5V $\pm 10\%$	5V $\pm 10\%$	5V $\pm 10\%$	5V $\pm 10\%$
Supply Current (I _{CC}) max.	30 mA	40 mA	85 mA	50 mA
OUTPUT				
	HS-100		HS-200	HS-500
	3.5 MHz-20 MHz	20+ MHz-30 MHz	225 KHz-4.0 MHz	25 MHz-60 MHz
V _{OH} (Logic "1")	+2.4V min. ¹	+2.7V min. ²	+2.4V min. ¹	+2.7V min. ²
V _{OL} (Logic "0")	+0.4V max. ³	+0.5V max. ⁴	+0.4V max. ³	+0.5V max. ⁴
Symmetry	60/40% ⁵	60/40% ⁵	55/45% ⁵	60/40% ⁵
T _R , T _F (Rise & Fall Time)	< 10 ns ⁶	< 5 ns ⁶	< 15 ns ⁶	< 5 ns ⁶
Output Short Circuit Current	18 mA min.	40 mA min.	18 mA min.	40 mA min.
Output Load	1 to 10 TTL Loads ⁷	1 to 10 TTL Loads ⁸	1 to 10 TTL Loads ⁷	1 to 10 TTL Loads ⁸
CONDITIONS				
¹ I _O source = -400 μA max.		⁴ I _O sink = 20.00 mA max.		⁷ 1.6 mA per load
² I _O source = -1.0 mA max.		⁵ V _O = 1.4V		⁸ 2.0 mA per load
³ I _O sink = 16.0 mA max.		⁶ (0.4V to 2.4V)		

Figure 10. Pre-Packaged Oscillator Data*

Prepackaged oscillators are available from most crystal manufacturers, and have the advantage that the system designer can treat the oscillator as a black box whose performance is guaranteed by people who carry many years of experience in designing and building oscillators. Figure 10 shows a typical data sheet for some prepackaged oscillators. Oscillators are also available with complementary outputs.

If the oscillator is to drive the microcontroller directly, one will want to make a careful comparison between the external drive requirements in the microcontroller data sheet and the oscillator's output logic levels and test conditions.

If oscillator stability is less critical than cost, the user may prefer to go with an in-house design. Not without some precautions, however.

It's easy to design oscillators that work. Almost all of them do work, even if the designer isn't too clear on why. The key point here is that *almost* all of them work. The problems begin when the system goes into production, and marginal units commence malfunctioning in the field. Most digital designers, after all, are not very adept at designing oscillators for production.

Oscillator design is somewhat of a black art, with the quality of the finished product being very dependent on the designer's experience and intuition. For that reason the most important consideration in any design is to have an adequate preproduction test program. Preproduction tests are discussed later in this Application Note. Here we will discuss some of the design options and take a look at some commonly used configurations.

Gate Oscillators versus Discrete Devices

Digital systems designers are understandably reluctant to get involved with discrete devices and their peculiarities (biasing techniques, etc.). Besides, the component count for these circuits tends to be quite a bit higher than what a digital designer is used to seeing for that amount of functionality. Nevertheless, if there are unusual requirements on the accuracy and stability of the clock frequency, it should be noted that discrete device oscillators can be tailored to suit the exact needs of the application and perfected to a level that would be difficult for a gate oscillator to approach.

In most cases, when an external oscillator is needed, the designer tends to rely on some form of a gate oscillator. A TTL inverter with a resistor connecting the output to the input makes a suitable inverting amplifier. The resistor holds the inverter in the transition region between logical high and low, so that at least for start-up purposes the inverter is a linear amplifier.

The feedback resistance has to be quite low, however, since it must conduct current sourced by the input pin without allowing the DC input voltage to get too far above the DC output voltage. For biasing purposes, the feedback resistance should not exceed a few k-ohms. But shunting the crystal with such a low resistance does not encourage start-up.

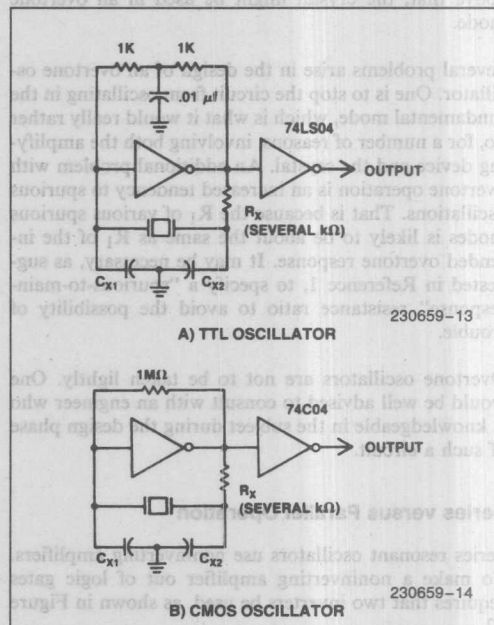


Figure 11. Commonly Used Gate Oscillators

Consequently, the configuration in Figure 11A might be suggested. By breaking R_f into two parts and AC-grounding the midpoint, one achieves the DC feedback required to hold the inverter in its active region, but without the negative signal feedback that is in effect telling the circuit *not* to oscillate. However, this biasing scheme will increase the start-up time, and relaxation-type oscillations are also possible.

A CMOS inverter, such as the 74HC04, might work better in this application, since a larger R_f can be used to hold the inverter in its linear region.

Logic gates tend to have a fairly low output resistance, which destabilizes the oscillator. For that reason a resistor R_x is often added to the feedback network, as shown in Figures 11A and B. At higher frequencies a 20 or 30 pF capacitor is sometimes used in the R_x position, to compensate for some of the internal propagation delay.

Reference 1 contains an excellent discussion of gate oscillators, and a number of design examples.

Fundamental versus Overtone Operation

It's easier to design an oscillator circuit to operate in the resonator's fundamental response mode than to design one for overtone operation. A quartz crystal whose fundamental response mode covers the desired frequency can be obtained up to some 30 MHz. For frequencies above that, the crystal might be used in an overtone mode.

Several problems arise in the design of an overtone oscillator. One is to stop the circuit from oscillating in the fundamental mode, which is what it would really rather do, for a number of reasons, involving both the amplifying device and the crystal. An additional problem with overtone operation is an increased tendency to spurious oscillations. That is because the R_1 of various spurious modes is likely to be about the same as R_1 of the intended overtone response. It may be necessary, as suggested in Reference 1, to specify a "spurious-to-main-response" resistance ratio to avoid the possibility of trouble.

Overtone oscillators are not to be taken lightly. One would be well advised to consult with an engineer who is knowledgeable in the subject during the design phase of such a circuit.

Series versus Parallel Operation

Series resonant oscillators use noninverting amplifiers. To make a noninverting amplifier out of logic gates requires that two inverters be used, as shown in Figure 12.

This type of circuit tends to be inaccurate and unstable in frequency over variations in temperature and V_{CC} . It has a tendency to oscillate at overtones, and to oscillate through C_0 of the crystal or some stray capacitance rather than as controlled by the mechanical resonance of the crystal.

The demon in series resonant oscillators is the phase shift in the amplifier. The series resonant oscillator wants more than just a "noninverting" amplifier—it wants a zero phase-shift amplifier. Multistage noninverting amplifiers tend to have a considerably lagging phase shift, such that the crystal reactance must be capacitive in order to bring the total phase shift around the feedback loop back up to 0. In this mode, a "12 MHz" crystal may be running at 8 or 9 MHz. One can put a capacitor in series with the crystal to relieve the crystal of having to produce all of the required phase shift, and bring the oscillation frequency closer to fs. However, to further complicate the situation, the amplifier's phase shift is strongly dependent on frequency, temperature, V_{CC} , and device sample.

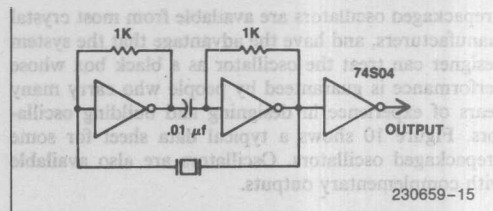


Figure 12. "Series Resonant" Gate Oscillator

Positive reactance oscillators ("parallel resonant") use inverting amplifiers. A single logic inverter can be used for the amplifier, as in Figure 11. The amplifier's phase shift is less critical, compared to a series resonant circuit, and since only one inverter is involved there's less phase error anyway. The oscillation frequency is effectively bounded by the resonant and antiresonant frequencies of the crystal itself. In addition, the feedback network includes capacitors that parallel the input and output terminals of the amplifier, thus reducing the effect of unpredictable capacitances at these points.

MORE ABOUT USING THE "ON-CHIP" OSCILLATORS

In this section we will describe the on-chip inverters on selected microcontrollers in some detail, and discuss criteria for selecting components to work with them. Future data sheets will supplement this discussion with updates and information pertinent to the use of each chip's oscillator circuitry.

Oscillator Calculations

Oscillator design, though aided by theory, is still largely an empirical exercise. The circuit is inherently nonlinear, and the normal analysis parameters vary with instantaneous voltage. In addition, when dealing with the on-chip circuitry, we have FETs being used as resistors, resistors being used as interconnects, distributed delays, input protection devices, parasitic junctions, and processing variations.

Consequently, oscillator calculations are never very precise. They can be useful, however, if they will at least indicate the effects of variations in the circuit parameters on start-up time, oscillation frequency, and steady-state amplitude. Start-up time, for example, can be taken as an indication of start-up reliability. If preproduction tests indicate a possible start-up problem, a relatively inexperienced designer can at least be made aware of what parameter may be causing the marginality, and what direction to go in to fix it.

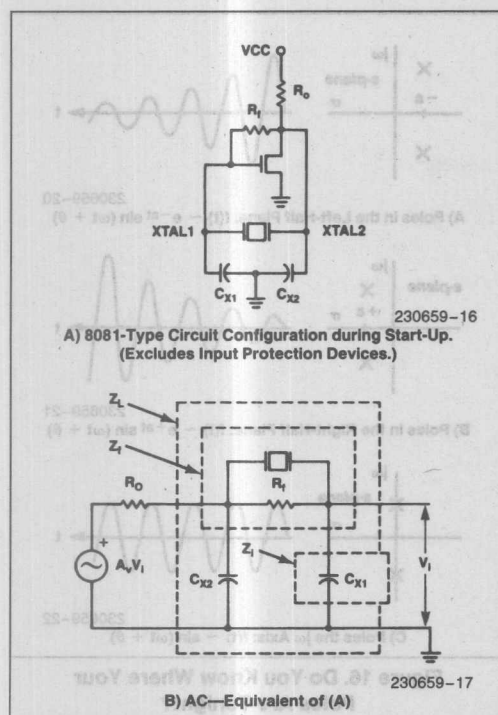


Figure 13. Oscillator Circuit Model Used in Start-Up Calculations

The analysis used here is mathematically straightforward but algebraically intractable. That means it's relatively easy to understand and program into a computer, but it will not yield a neat formula that gives, say, steady-state amplitude as a function of this or that list of parameters. A listing of a BASIC program that implements the analysis will be found in Appendix II.

When the circuit is first powered up, and before the oscillations have commenced (and if the oscillations fail to commence), the oscillator can be treated as a small signal linear amplifier with feedback. In that case, standard small-signal analysis techniques can be used to determine start-up characteristics. The circuit model used in this analysis is shown in Figure 13.

The circuit approximates that there are no high-frequency effects within the amplifier itself, such that its high-frequency behavior is dominated by the load impedance Z_L . This is a reasonable approximation for single-stage amplifiers of the type used in 8051-type devices. Then the gain of the amplifier as a function of frequency is

$$A = \frac{A_v Z_L}{Z_L + R_0}$$

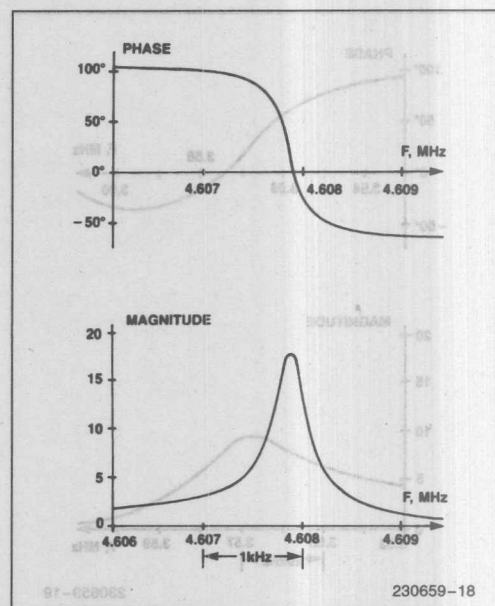


Figure 14. Loop Gain versus Frequency (4.608 MHz Crystal)

The gain of the feedback network is

$$\beta = \frac{Z_f}{Z_f + Z_L}$$

And the loop gain is

$$\beta A = \frac{Z_f}{Z_f + Z_L} \times \frac{A_v Z_L}{Z_L + R_0}$$

The impedances Z_L , Z_f , and Z_i are defined in Figure 13B.

Figure 14 shows the way the loop gain thus calculated (using typical 8051-type parameters and a 4.608 MHz crystal) varies with frequency. The frequency of interest is the one for which the phase of the loop gain is zero. The accepted criterion for start-up is that the magnitude of the loop gain must exceed unity at this frequency. This is the frequency at which the circuit is in resonance. It corresponds very closely with the antiresonant frequency of the motional arm of the crystal in parallel with C_L .

Figure 15 shows the way the loop gain varies with frequency when the parameters of a 3.58 MHz ceramic resonator are used in place of a crystal (the amplifier parameters being typical 8051, as in Figure 14). Note the different frequency scales.

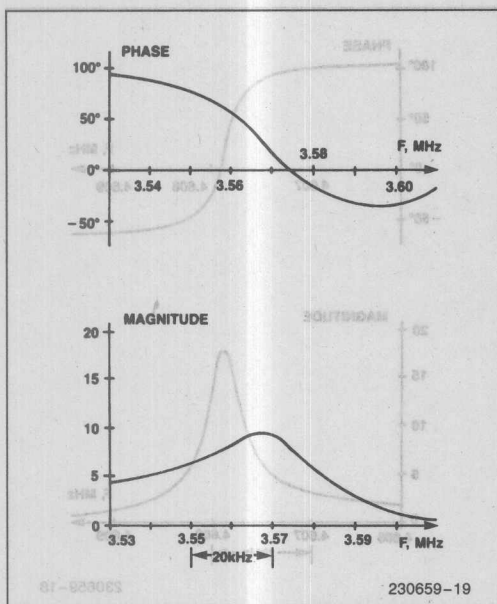


Figure 15. Loop Gain versus Frequency (3.58 MHz Ceramic)

Start-Up Characteristics

It is common, in studies of feedback systems, to examine the behavior of the closed loop gain as a function of complex frequency $s = \sigma + j\omega$; specifically, to determine the location of its poles in the complex plane. A pole is a point on the complex plane where the gain function goes to infinity. Knowledge of its location can be used to predict the response of the system to an input disturbance.

The way that the response function depends on the location of the poles is shown in Figure 16. Poles in the left-half plane cause the response function to take the form of a damped sinusoid. Poles in the right-half plane cause the response function to take the form of an exponentially growing sinusoid. In general,

$$v(t) \sim e^{at} \sin(\omega t + \theta)$$

where a is the real part of the pole frequency. Thus if the pole is in the right-half plane, a is positive and the sinusoid grows. If the pole is in the left-half plane, a is negative and the sinusoid is damped.

The same type of analysis can usefully be applied to oscillators. In this case, however, rather than trying to ensure that the poles are in the left-half plane, we would seek to ensure that they're in the *right*-half plane. An exponentially growing sinusoid is exactly what is wanted from an oscillator that has just been powered up.

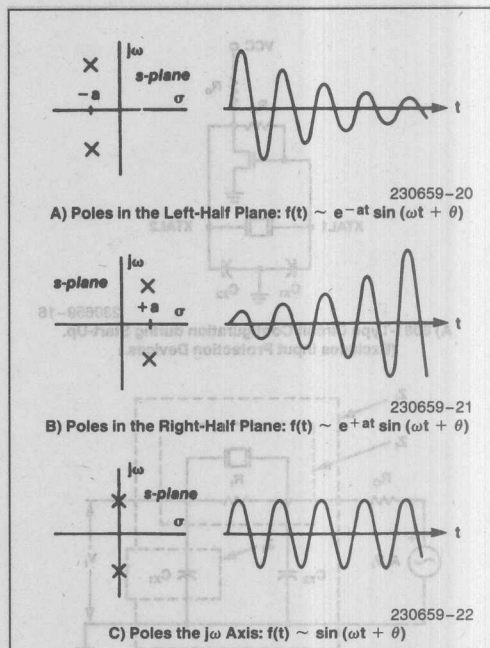


Figure 16. Do You Know Where Your Poles Are Tonight?

The gain function of interest in oscillators is $1/(1 - \beta A)$. Its poles are at the complex frequencies where $\beta A = 1 \angle 0^\circ$, because that value of βA causes the gain function to go to infinity. The oscillator will start up if the real part of the pole frequency is positive. More importantly, the *rate* at which it starts up is indicated by how *much* greater than 0 the real part of the pole frequency is.

The circuit in Figure 13B can be used to find the pole frequencies of the oscillator gain function. All that needs to be done is evaluate the impedances at complex frequencies $\sigma + j\omega$ rather than just at ω , and find the value of $\sigma + j\omega$ for which $\beta A = 1 \angle 0^\circ$. The larger that value of σ is, the faster the oscillator will start up.

Of course, other things besides pole frequencies, things like the VCC rise time, are at work in determining the start-up time. But to the extent that the pole frequencies *do* affect start-up time, we can obtain results like those in Figures 17 and 18.

To obtain these figures the pole frequencies were computed for various values of capacitance C_X from XTAL1 and XTAL2 to ground (thus $C_{X1} = C_{X2} = C_X$). Then a "time constant" for start-up was calculated

as $T_s = \frac{1}{\sigma}$ where σ is the real part of the pole frequency (rad/sec), and this time constant is plotted versus C_X .

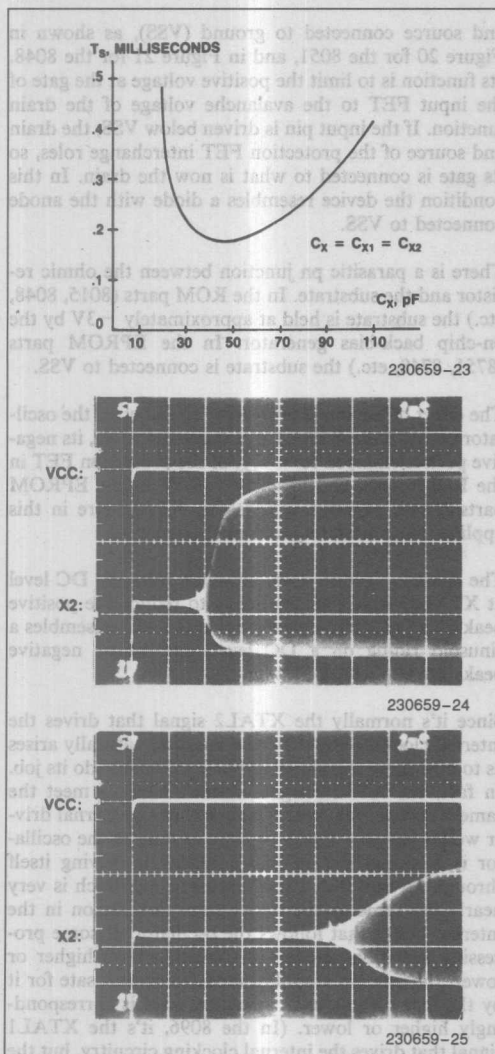


Figure 17. Oscillator Start-Up (4.608 MHz Crystal from Standard Crystal Corp.)

A short time constant means faster start-up. A long time constant means slow start-up. Observations of actual start-ups are shown in the figures. Figure 17 is for a typical 8051 with a 4.608 MHz crystal supplied by Standard Crystal Corp., and Figure 18 is for a typical 8051 with a 3.58 MHz ceramic resonator supplied by NTK Technical Ceramics, Ltd.

It can be seen in Figure 17 that, for this crystal, values of C_X between 30 and 50 pF minimize start-up time, but that the exact value in this range is not particularly important, even if the start-up time itself is critical.

As previously mentioned, start-up time can be taken as an indication of start-up reliability. Start-up problems are normally associated with C_{X1} and C_{X2} being too small or too large for a given resonator. If the parameters of the resonator are known, curves such as in Figure 17 or 18 can be generated to define acceptable ranges of values for these capacitors.

As the oscillations grow in amplitude, they reach a level at which they undergo severe clipping within the amplifier, in effect reducing the amplifier gain. As the amplifier gain decreases, the poles move towards the $j\omega$ axis. In steady-state, the poles are on the $j\omega$ axis and the amplitude of the oscillations is constant.

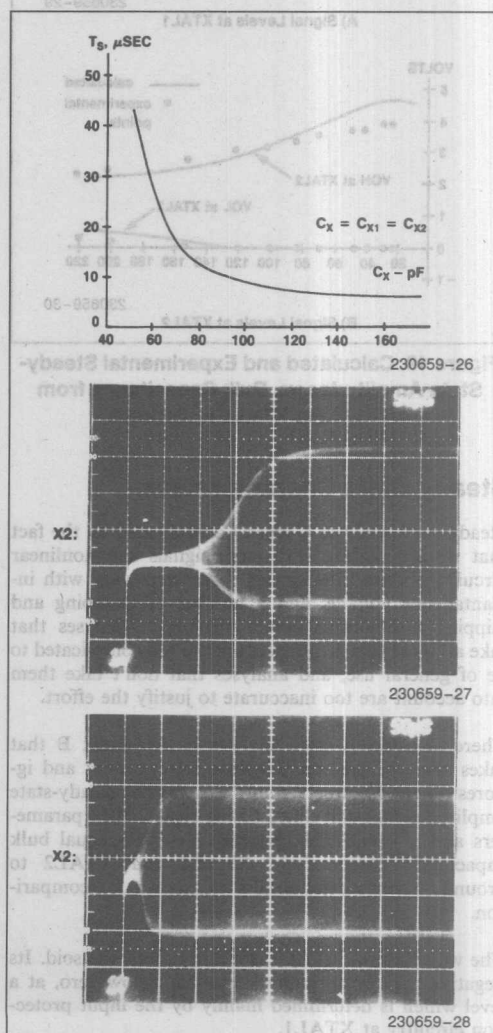


Figure 18. Oscillator Start-Up (3.58 MHz Ceramic Resonator from NTK Technical Ceramics)

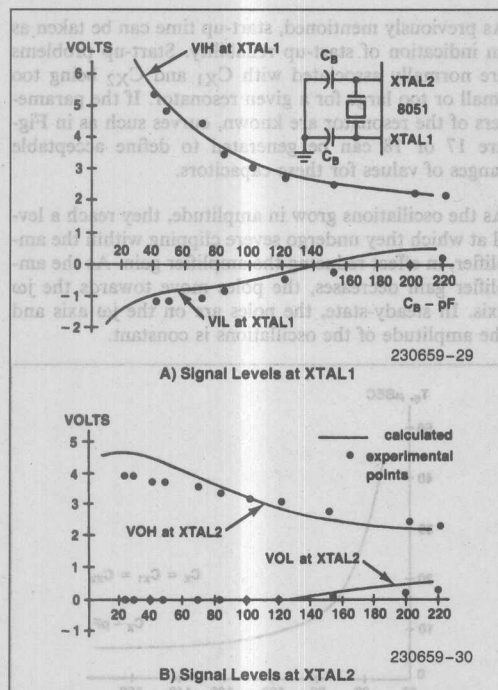


Figure 19. Calculated and Experimental Steady-State Amplitudes vs. Bulk Capacitance from XTAL1 and XTAL2 to Ground

Steady-State Characteristics

Steady-state analysis is greatly complicated by the fact that we are dealing with large signals and nonlinear circuit response. The circuit parameters vary with instantaneous voltage, and a number of clamping and clipping mechanisms come into play. Analyses that take all these things into account are too complicated to be of general use, and analyses that don't take them into account are too inaccurate to justify the effort.

There is a steady-state analysis in Appendix B that takes some of the complications into account and ignores others. Figure 19 shows the way the steady-state amplitudes thus calculated (using typical 8051 parameters and a 4.608 MHz crystal) vary with equal bulk capacitance placed from XTAL1 and XTAL2 to ground. Experimental results are shown for comparison.

The waveform at XTAL1 is a fairly clean sinusoid. Its negative peak is normally somewhat below zero, at a level which is determined mainly by the input protection circuitry at XTAL1.

The input protection circuitry consists of an ohmic resistor and an enhancement-mode FET with the gate

and source connected to ground (VSS), as shown in Figure 20 for the 8051, and in Figure 21 for the 8048. Its function is to limit the positive voltage at the gate of the input FET to the avalanche voltage of the drain junction. If the input pin is driven below VSS, the drain and source of the protection FET interchange roles, so its gate is connected to what is now the drain. In this condition the device resembles a diode with the anode connected to VSS.

There is a parasitic pn junction between the ohmic resistor and the substrate. In the ROM parts (8015, 8048, etc.) the substrate is held at approximately $-3V$ by the on-chip back-bias generator. In the EPROM parts (8751, 8748, etc.) the substrate is connected to VSS.

The effect of the input protection circuitry on the oscillator is that if the XTAL1 signal goes negative, its negative peak is clamped to $-V_{DS}$ of the protection FET in the ROM parts, and to about $-0.5V$ in the EPROM parts. These negative voltages on XTAL1 are in this application self-limiting and nondestructive.

The clamping action does, however, raise the DC level at XTAL1, which in turn tends to reduce the positive peak at XTAL2. The waveform at XTAL2 resembles a sinusoid riding on a DC level, and whose negative peaks are clipped off at zero.

Since it's normally the XTAL2 signal that drives the internal clocking circuitry, the question naturally arises as to how large this signal must be to reliably do its job. In fact, the XTAL2 signal doesn't have to meet the same V_{IH} and V_{IL} specifications that an external driver would have to. That's because as long as the oscillator is working, the on-chip amplifier is driving itself through its own 0-to-1 transition region, which is very nearly the same as the 0-to-1 transition region in the internal buffer that follows the oscillator. If some processing variations move the transition level higher or lower, the on-chip amplifier tends to compensate for it by the fact that its own transition level is correspondingly higher or lower. (In the 8096, it's the XTAL1 signal that drives the internal clocking circuitry, but the same concept applies.)

The main concern about the XTAL2 signal amplitude is an indication of the general health of the oscillator. An amplitude of less than about 2.5V peak-to-peak indicates that start-up problems could develop in some units (with low gain) with some crystals (with high R_1). The remedy is to either adjust the values of C_{X1} and/or C_{X2} or use a crystal with a lower R_1 .

The amplitudes at XTAL1 and XTAL2 can be adjusted by changing the ratio of the capacitors from XTAL1 and XTAL2 to ground. Increasing the XTAL2 capacitance, for example, decreases the amplitude at XTAL2 and increases the amplitude at XTAL1 by about the same amount. Decreasing both caps increases both amplitudes.

Pin Capacitance

Internal pin-to-ground and pin-to-pin capacitances at XTAL1 and XTAL2 will have some effect on the oscillator. These capacitances are normally taken to be in the range of 5 to 10 pF, but they are extremely difficult to evaluate. Any measurement of one such capacitance will necessarily include effects from the others. One advantage of the positive reactance oscillator is that the pin-to-ground capacitances are paralleled by external bulk capacitors, so a precise determination of their value is unnecessary. We would suggest that there is little justification for more precision than to assign them a value of 7 pF (XTAL1-to-ground and XTAL1-to-XTAL2). This value is probably not in error by more than 3 or 4 pF.

The XTAL2-to-ground capacitance is not entirely "pin capacitance," but more like an "equivalent output capacitance" of some 25 to 30 pF, having to include the effect of internal phase delays. This value will vary to some extent with temperature, processing, and frequency.

MCS®-51 Oscillator

The on-chip amplifier on the HMOS MCS-51 family is shown in Figure 20. The drain load and feedback "resistors" are seen to be field-effect transistors. The drain load FET, R_D , is typically equivalent to about 1K to 3 K-ohms. As an amplifier, the low frequency voltage gain is normally between -10 and -20, and the output resistance is effectively R_D .

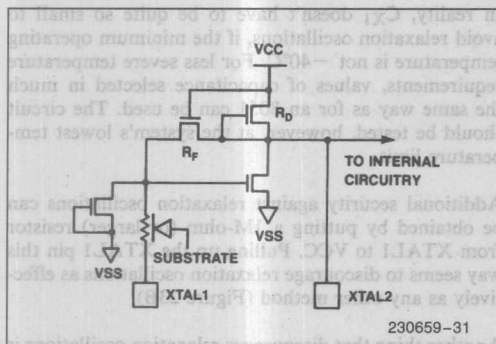


Figure 20. MCS®-51 Oscillator Amplifier

The 80151 oscillator is normally used with equal bulk capacitors placed externally from XTAL1 to ground and from XTAL2 to ground. To determine a reasonable value of capacitance to use in these positions, given a crystal of ceramic resonator of known parameters, one can use the BASIC analysis in Appendix II to generate curves such as in Figures 17 and 18. This procedure will define a range of values that will minimize start-up time. We don't suggest that smaller values be

used than those which minimize start-up time. Larger values than those can be used in applications where increased frequency stability is desired, at some sacrifice in start-up time.

Standard Crystal Corp. (Reference 8) studied the use of their crystals with the MCS-51 family using skew sample supplied by Intel. They suggest putting 30 pF capacitors from XTAL1 and XTAL2 to ground, if the crystal is specified as described in Reference 8. They noted that in that configuration and with crystals thus specified, the frequency accuracy was $\pm 0.01\%$ and the frequency stability was $\pm 0.005\%$, and that a frequency accuracy of $\pm 0.005\%$ could be obtained by substituting a 25 pF fixed cap in parallel with a 5-20 pF trimmer for one of the 30 pF caps.

MCS-51 skew samples have also been supplied to a number of ceramic resonator manufacturers for characterization with their products. These companies should be contacted for application information on their products. In general, however, ceramics tend to want somewhat larger values for C_{X1} and C_{X2} than quartz crystals do. As shown in Figure 18, they start up a lot faster that way.

In some application the actual frequency tolerance required is only 1% or so, the user being concerned mainly that the circuit will oscillate. In that case, C_{X1} and C_{X2} can be selected rather freely in the range of, 20 to 80 pF.

As you can see, "best" values for these components and their tolerances are strongly dependent on the application and its requirements. In any case, their suitability should be verified by environmental testing before the design is submitted to production.

MCS®-48 Oscillator

The NMOS and HMOS MCS-48 oscillator is shown in Figure 21. It differs from the 8051 in that its inverting

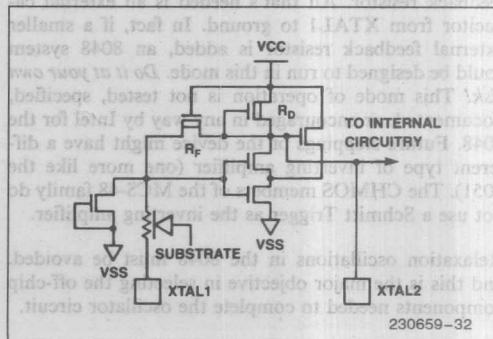


Figure 21. MCS®-48 Oscillator Amplifier

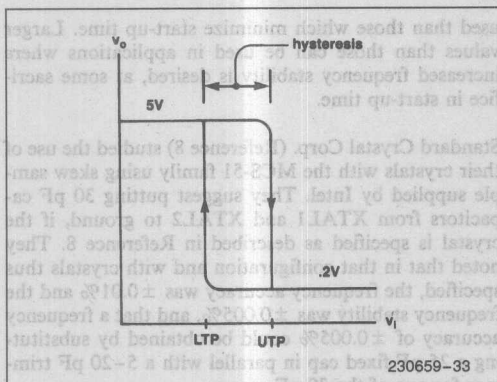


Figure 22. Schmitt Trigger Characteristic

amplifier is a Schmitt Trigger. This configuration was chosen to prevent crosstalk from the TO pin, which is adjacent to the XTAL1 pin.

All Schmitt Trigger circuits exhibit a hysteresis effect, as shown in Figure 22. The hysteresis is what makes it less sensitive to noise. The same hysteresis allows any Schmitt Trigger to be used as a relaxation oscillator. All you have to do is connect a resistor from output to input, and a capacitor from input to ground, and the circuit oscillates in a relaxation mode as follows.

If the Schmitt Trigger output is at a logic high, the capacitor commences charging through the feedback resistor. When the capacitor voltage reaches the upper trigger point (UTP), the Schmitt Trigger output switches to a logic low and the capacitor commences discharging through the same resistor. When the capacitor voltage reaches the lower trigger point (LTP), the Schmitt Trigger output switches to a logic high again, and the sequence repeats. The oscillation frequency is determined by the RC time constant and the hysteresis voltage, UTP-LTP.

The 8048 can oscillate in this mode. It has an internal feedback resistor. All that's needed is an external capacitor from XTAL1 to ground. In fact, if a smaller external feedback resistor is added, an 8048 system could be designed to run in this mode. *Do it at your own risk!* This mode of operation is not tested, specified, documented, or encouraged in any way by Intel for the 8048. Future steppings of the device might have a different type of inverting amplifier (one more like the 8051). The CHMOS members of the MCS-48 family do not use a Schmitt Trigger as the inverting amplifier.

Relaxation oscillations in the 8048 must be avoided, and this is the major objective in selecting the off-chip components needed to complete the oscillator circuit.

When an 8048 is powered up, if VCC has a short rise time, the relaxation mode starts first. The frequency is normally about 50 KHz. The resonator mode builds

more slowly, but it eventually takes over and dominates the operation of the circuit. This is shown in Figure 23A. Due to processing variations, some units seem to have a harder time coming out of the relaxation mode, particularly at low temperatures. In some cases the resonator oscillations may fail entirely, and leave the device in the relaxation mode. Most units will stick in the relaxation mode at any temperature if C_{X1} is larger than about 50 pF. Therefore, C_{X1} should be chosen with some care, particularly if the system must operate at lower temperatures.

One method that has proven effective in all units to -40°C is to put 5 pF from XTAL1 to ground and 20 pF from XTAL2 to ground. Unfortunately, while this method does discourage the relaxation mode, it is not an optimal choice for the resonator mode. For one thing, it does not swamp the pin capacitance. Also, it makes for a rather high signal level at XTAL1 (8 or 9 volts peak-to-peak).

The question arises as to whether that level of signal at XTAL1 might damage the chip. Not to worry. The negative peaks are self-limiting and nondestructive. The positive peaks could conceivably damage the oxide, but in fact, NMOS chips (eg, 8048) and HMOS chips (eg, 8048H) are tested to a much higher voltage than that. The technology trend, of course, is to thinner oxides, as the devices shrink in size. For an extra margin of safety, the HMOS II chips (eg, 8048AH) have an internal diode clamp at XTAL1 to VCC.

In reality, C_{X1} doesn't have to be quite so small to avoid relaxation oscillations, if the minimum operating temperature is not -40°C . For less severe temperature requirements, values of capacitance selected in much the same way as for an 8051 can be used. The circuit should be tested, however, at the system's lowest temperature limit.

Additional security against relaxation oscillations can be obtained by putting a 1M-ohm (or larger) resistor from XTAL1 to VCC. Pulling up the XTAL1 pin this way seems to discourage relaxation oscillations as effectively as any other method (Figure 23B).

Another thing that discourages relaxation oscillations is low VCC. The resonator mode, on the other hand is much less sensitive to VCC. Thus if VCC comes up relatively slowly (several milliseconds rise time), the resonator mode is normally up and running before the relaxation mode starts (in fact, before VCC has even reached operating specs). This is shown in Figure 23C.

A secondary effect of the hysteresis is a shift in the oscillation frequency. At low frequencies, the output signal from an inverter without hysteresis leads (or lags) the input by 180 degrees. The hysteresis in a Schmitt Trigger, however, causes the output to lead the

input by less than 180 degrees (or lag by more than 180 degrees), by an amount that depends on the signal amplitude, as shown in Figure 24. At higher frequencies, there are additional phase shifts due to the various reactances in the circuit, but the phase shift due to the hysteresis is still present. Since the total phase shift in the oscillator's loop gain is necessarily 0 or 360 degrees, it is apparent that as the oscillations build up, the frequency has to change to allow the reactances to compensate for the hysteresis. In normal operation, this additional phase shift due to hysteresis does not exceed a few degrees, and the resulting frequency shift is negligible.

Kyocera, a ceramic resonator manufacturer, studied the use of some of their resonators (at 6.0 MHz, 8.0 MHz, and 11.0 MHz) with the 8049H. Their conclusion as to the value of capacitance to use at XTAL1 and XTAL2 was that 33 pF is appropriate at all three frequencies. One should probably follow the manufacturer's recommendations in this matter, since they will guarantee operation.

Whether one should accept these recommendations and guarantees without further testing is, however, another matter. Not all users have found the recommendations to be without occasional problems. If you run into diffi-

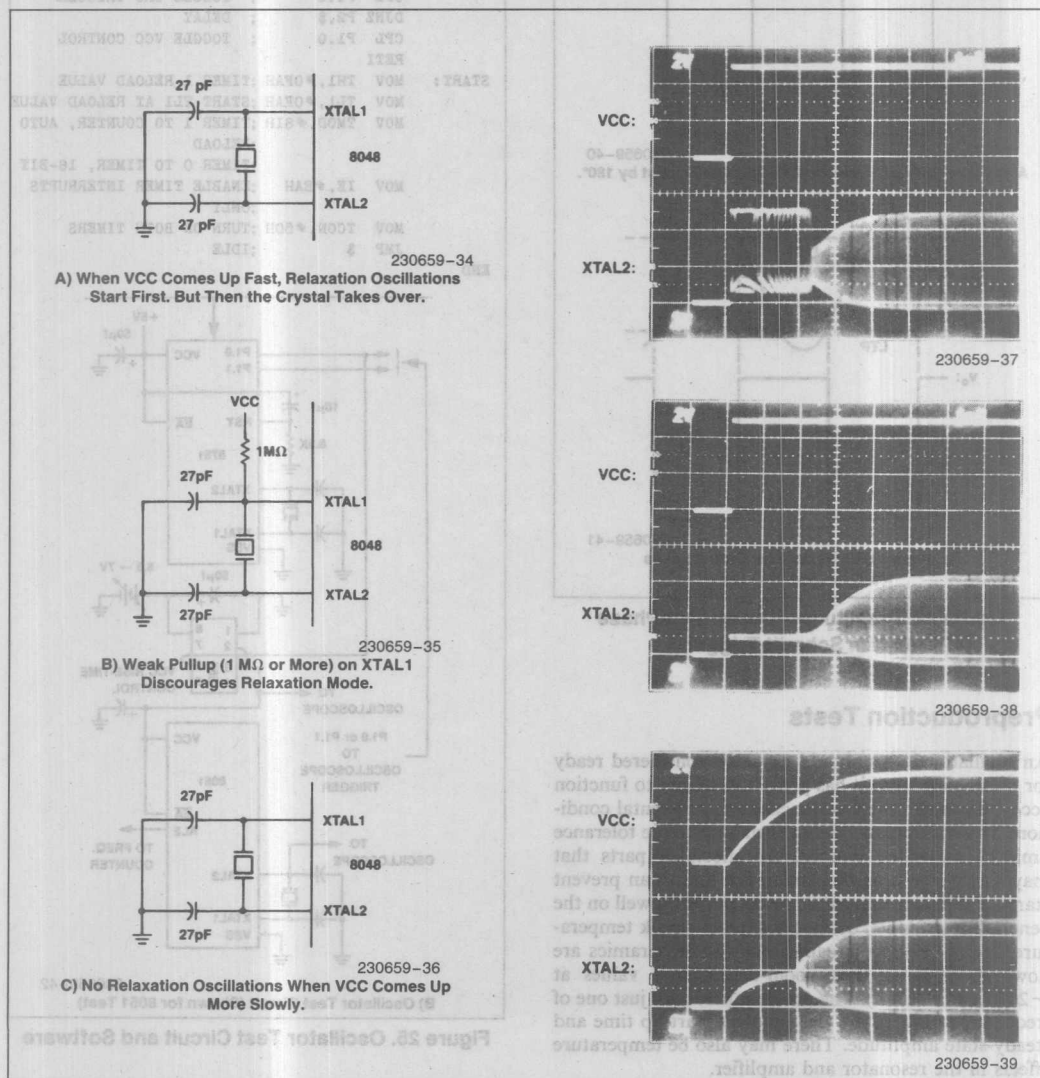


Figure 23. Relaxation Oscillations in the 8048

culties using their recommendations, both Intel and the ceramic resonator manufacturer want to know about it. It is to their interest, and ours, that such problems be resolved.

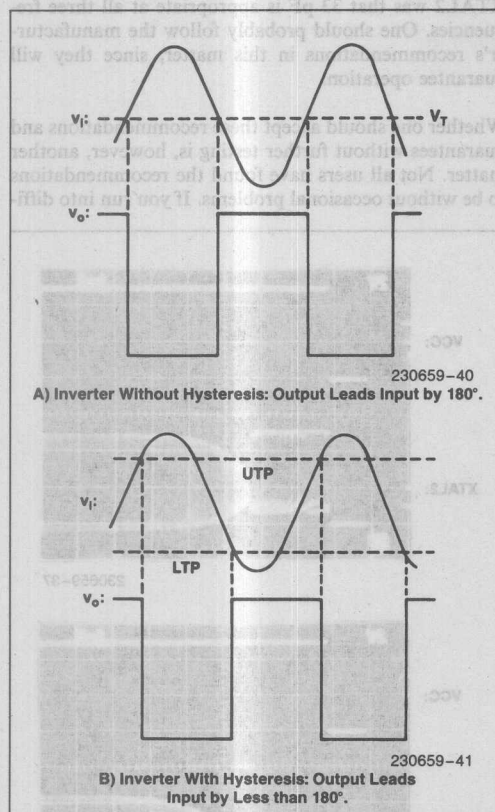


Figure 24. Amplitude-Dependent Phase Shift in Schmitt Trigger

Preproduction Tests

An oscillator design should never be considered ready for production until it has proven its ability to function acceptably well under worst-case environmental conditions and with parameters at their worst-case tolerance limits. Unexpected temperature effects in parts that may already be near their tolerance limits can prevent start-up of an oscillator that works perfectly well on the bench. For example, designers often overlook temperature effects in ceramic capacitors. (Some ceramics are down to 50% of their room-temperature values at -20°C and $+60^{\circ}\text{C}$). The problem here isn't just one of frequency stability, but also involves start-up time and steady-state amplitude. There may also be temperature effects in the resonator and amplifier.

It will be helpful to build a test jig that will allow the oscillator circuit to be tested independently of the rest of the system. Both start-up and steady-state characteristics should be tested. Figure 25 shows the circuit that

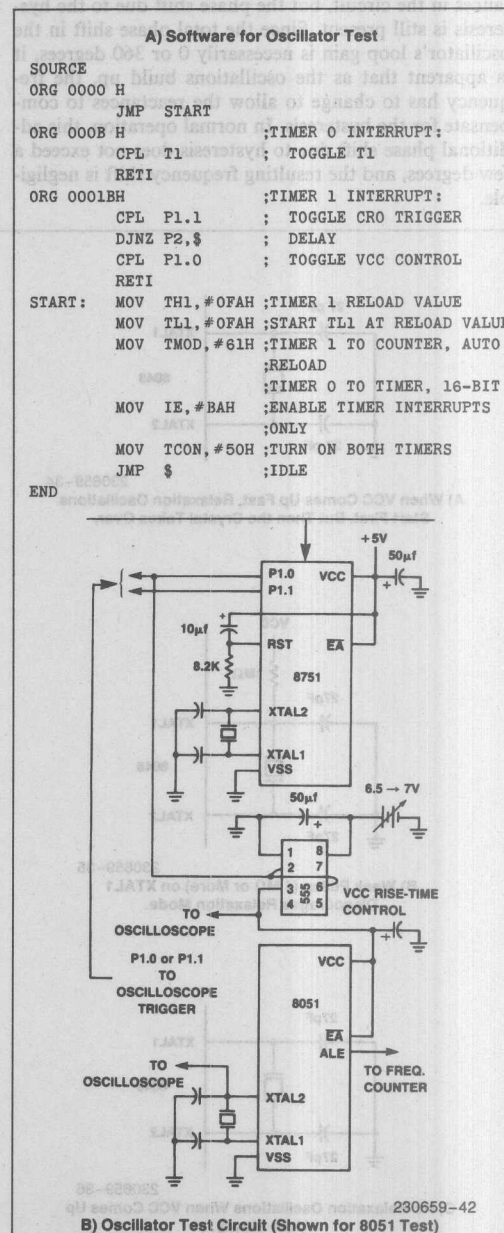


Figure 25. Oscillator Test Circuit and Software

was used to obtain the oscillator start-up photographs in this Application Note. This circuit or a modified version of it would make a convenient test vehicle. The oscillator and its relevant components can be physically separated from the control circuitry, and placed in a temperature chamber.

Start-up should be observed under a variety of conditions, including low VCC and using slow and fast VCC rise times. The oscillator should not be reluctant to start-up even when VCC is below its spec value for the rest of the chip. (The rest of the chip may not function, but the oscillator should work.) It should also be verified that start-up occurs when the resonator has more than its upper tolerance limit of series resistance. (Put some resistance in series with the resonator for this test.) The bulk capacitors from XTAL1 and XTAL2 to ground should also be varied to their tolerance limits.

The same circuit, with appropriate changes in the software to lengthen the "on" time, can be used to test the steady-state characteristics of the oscillator, specifically the frequency, frequency stability, and amplitudes at XTAL1 and XTAL2.

As previously noted, the voltage swings at these pins are not critical, but they should be checked at the system's temperature limits to ensure that they are in good health. Observing these signals necessarily changes them somewhat. Observing the signal at XTAL2 requires that the capacitor at that pin be reduced to account for the oscilloscope probe capacitance. Observing the signal at XTAL1 requires the same consideration, plus a blocking capacitor (switch the oscilloscope input to AC), so as to not disturb the DC level at that pin. Alternatively, a MOSFET buffer such as the one shown in Figure 26 can be used. It should be verified by direct measurement that the ground clip on the scope probe is ohmically connected to the scope chassis (probes are incredibly fragile in this respect), and the observations should be made with the ground clip on the VSS pin, or very close to it. If the probe shield isn't operational and in use, the observations are worthless.

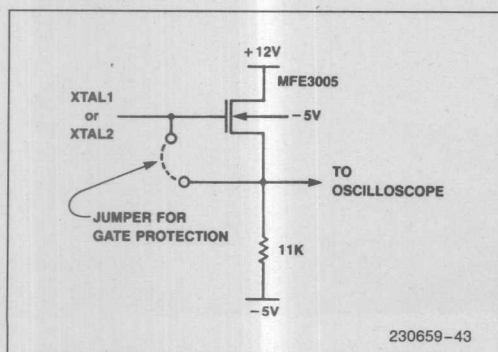


Figure 26. MOSFET Buffer for Observing Oscillator Signals

Frequency checks should be made with only the oscillator circuitry connected to XTAL1 and XTAL2. The ALE frequency can be counted, and the oscillator frequency derived from that. In systems where the frequency tolerance is only "nominal," the frequency should still be checked to ascertain that the oscillator isn't running in a spurious resonance or relaxation mode. Switching VCC off and on again repeatedly will help reveal a tendency to go into unwanted modes of oscillation.

The operation of the oscillator should then be verified under actual system running conditions. By this stage one will be able to have some confidence that the basic selection of components for the oscillator itself is suitable, so if the oscillator appears to malfunction in the system the fault is not in the selection of these components.

Troubleshooting Oscillator Problems

The first thing to consider in case of difficulty is that between the test jig and the actual application there may be significant differences in stray capacitances, particularly if the actual application is on a multi-layer board.

Noise glitches, that aren't present in the test jig but are in the application board, are another possibility. Capacitive coupling between the oscillator circuitry and other signal has already been mentioned as a source of miscounts in the internal clocking circuitry. Inductive coupling is also possible, if there are strong currents nearby. These problems are a function of the PCB layout.

Surrounding the oscillator components with "quiet" traces (VCC and ground, for example) will alleviate capacitive coupling to signals that have fast transition times. To minimize inductive coupling, the PCB layout should minimize the areas of the loops formed by the oscillator components. These are the loops that should be checked:

- XTAL1 through the resonator to XTAL2;
- XTAL1 through C_{X1} to the VSS pin;
- XTAL2 through C_{X2} to the VSS pin.

It is not unusual to find that the grounded ends of C_{X1} and C_{X2} eventually connect up to the VSS pin only after looping around the farthest ends of the board. Not good.

Finally, it should not be overlooked that software problems sometimes imitate the symptoms of a slow-starting oscillator or incorrect frequency. Never underestimate the perversity of a software problem.

REFERENCES

1. Frerking, M. E., *Crystal Oscillator Design and Temperature Compensation*, Van Nostrand Reinhold, 1978.
2. Bottom, V., "The Crystal Unit as a Circuit Component," Ch. 7, *Introduction to Quartz Crystal Unit Design*, Van Nostrand Reinhold, 1982.
3. Parzen, B., *Design of Crystal and Other Harmonic Oscillators*, John Wiley & Sons, 1983.
4. Holmbeck, J. D., "Frequency Tolerance Limitations with Logic Gate Clock Oscillators, 31st Annual Frequency Control Symposium, June, 1977.
5. Roberge, J. K., "Nonlinear Systems," Ch. 6, *Operational Amplifiers: Theory and Practice*, Wiley, 1975.
6. Eaton, S. S., *Timekeeping Advances Through COS/MOS Technology*, RCA Application Note ICAN-6086.
7. Eaton, S. S., *Micropower Crystal-Controlled Oscillator Design Using RCA COS/MOS Inverters*, RCA Application Note ICAN-6539.
8. Fisher, J. B., *Crystal Specifications for the Intel 8031/8051/8751 Microcontrollers*, Standard Crystal Corp. Design Data Note #2F.
9. Murata Mfg. Co., Ltd., *Ceramic Resonator "Ceralock" Application Manual*.
10. Kyoto Ceramic Co., Ltd., *Adaptability Test Between Intel 8049H and Kyocera Ceramic Resonators*.
11. Kyoto Ceramic Co., Ltd., *Technical Data on Ceramic Resonator Model KBR-6.0M, KBR-8.0M, KBR-11.0M Application for 8051 (Intel)*.
12. NTK Technical Ceramic Division, NGK Spark Plug Co., Ltd., *NTKK Ceramic Resonator Manual*.

Noise spikes that aren't present in the test jig but are in the application board, are another possibility. Capacitive coupling between the oscillator circuitry and other signal has already been mentioned as a source of miscounts in the internal clocking circuitry. Inductive coupling is also possible if there are strong currents nearby. These problems are a function of the PCB layout.

Surrounding the oscillator components with "quiet" traces (VCC and ground, for example) will alleviate capacitive coupling to signals that have fast transition times. To minimize inductive coupling, the PCB layout should minimize the areas of the loops formed by the oscillator components. These are the loops that should be checked:

XTAL1 through the resonator to XTAL2.
XTAL1 through C₁ to the VSS pin.
XTAL2 through C₂ to the VSS pin.

It is not unusual to find that the grounded ends of C₁ and C₂ eventually connect up to the VSS pin only after looking around the farthest ends of the board. Not good.

Finally, it should not be overlooked that software problems sometimes imitate the symptoms of a slow-starting oscillator or incorrect frequency. Never underestimate the pervasiveness of a software problem.

As previously noted, the voltage swings at these pins are not critical, but they should be checked at the system's temperature limits to ensure that they are in good health. Observing these signals necessarily changes them somewhat. Observing the signal at XTAL2 requires that the capacitor at that pin be reduced to account for the oscilloscope probe capacitance. Observing the signal at XTAL1 requires the same consideration, plus a blocking capacitor (switch the oscilloscope input to AC) so as not to disturb the DC level at that pin. Alternatively, a MOSFET buffer such as the one shown in Figure 26 can be used. It should be verified by direct measurement that the ground clip on the scope probe is electrically connected to the scope chassis (probes are incredibly fragile in this respect) and the observations should be made with the ground clip on the VSS pin or very close to it. If the probe shield isn't operational and in use, the observations are worthless.

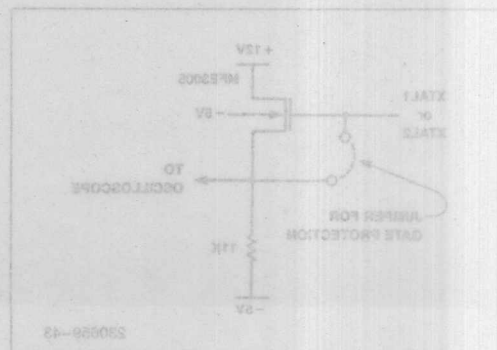


Figure 26. MOSFET Buffer for Observing Oscillator Signals

APPENDIX A QUARTZ AND CERAMIC RESONATOR FORMULAS

Based on the equivalent circuit of the crystal, the impedance of the crystal is

$$Z_{\text{XTAL}} = \frac{(R_1 + j\omega L_1 + 1/j\omega C_1)(1/j\omega C_0)}{R_1 + j\omega L_1 + 1/j\omega C_1 + 1/j\omega C_0}$$

After some algebraic manipulation, this calculation can be written in the form

$$Z_{\text{XTAL}} = \frac{1}{j\omega(C_1 + C_0)} \cdot \frac{1 - \omega^2 L_1 C_1 + j\omega R_1 C_1}{1 - \omega^2 L_1 C_T + j\omega R_1 C_T}$$

where C_T is the capacitance of C_1 in series with C_0 :

$$C_T = \frac{C_1 C_0}{C_1 + C_0}$$

The impedance of the crystal in parallel with an external load capacitance C_L is the same expression, but with $C_0 + C_L$ substituted for C_0 :

$$Z_{\text{XTAL} \parallel C_L} = \frac{1}{j\omega(C_1 + C_0 + C_L)} \cdot \frac{1 - \omega^2 L_1 C_1 + j\omega R_1 C_1}{1 - \omega^2 L_1 C'_T + j\omega R_1 C'_T}$$

where C'_T is the capacitance of C_1 in series with $(C_0 + C_L)$:

$$C'_T = \frac{C_1(C_0 + C_L)}{C_1 + C_0 + C_L}$$

The impedance of the crystal in series with the load capacitance is

$$\begin{aligned} Z_{\text{XTAL} + C_L} &= Z_{\text{XTAL}} + \frac{1}{j\omega C_L} \\ &= \frac{C_L + C_1 + C_0}{j\omega C_L(C_1 + C_0)} \cdot \frac{1 - \omega^2 L_1 C'_T + j\omega R_1 C'_T}{1 - \omega^2 L_1 C_T + j\omega R_1 C_T} \end{aligned}$$

where C_T and C'_T are as defined above.

The phase angles of these impedances are readily obtained from the impedance expressions themselves:

$$\begin{aligned} \theta_{\text{XTAL}} &= \arctan \frac{\omega R_1 C_1}{1 - \omega^2 L_1 C_1} \\ &\quad - \arctan \frac{\omega R_1 C_T}{1 - \omega^2 L_1 C_T} - \frac{\pi}{2} \end{aligned}$$

$$\theta_{\text{XTAL} \parallel C_L} = \arctan \frac{\omega R_1 C_1}{1 - \omega^2 L_1 C_1}$$

$$- \arctan \frac{\omega R_1 C'_T}{1 - \omega^2 L_1 C'_T} - \frac{\pi}{2}$$

$$\theta_{\text{XTAL} + C_L} = \arctan \frac{\omega R_1 C'_T}{1 - \omega^2 L_1 C'_T}$$

The resonant ("series resonant") frequency is the frequency at which the phase angle is zero and the impedance is low. The antiresonant ("parallel resonant") frequency is the frequency at which the phase angle is zero and the impedance is high.

Each of the above θ -expressions contains two arctan functions. Setting the denominator of the argument of the first arctan function to zero gives (approximately) the "series resonant" frequency for that configuration. Setting the denominator of the argument of the second arctan function to zero gives (approximately) the "parallel resonant" frequency for that configuration.

For example, the resonant frequency of the crystal is the frequency at which

$$1 - \omega^2 L_1 C_1 = 0$$

Thus

$$\omega_s = \frac{1}{\sqrt{L_1 C_1}}$$

or

$$f_s = \frac{1}{2\pi\sqrt{L_1 C_1}}$$

It will be noted that the series resonant frequency of the "XTAL + CL" configuration (crystal in series with CL) is the same as the parallel resonant frequency of the "XTAL||CL" configuration (crystal in parallel with CL). This is the frequency at which

$$1 - \omega^2 L_1 C_T = 0$$

Thus

$$\omega_a = \frac{1}{\sqrt{L_1 C_T}}$$

or

$$f_a = \frac{1}{2\pi\sqrt{L_1 C_T}}$$

This fact is used by crystal manufacturers in the process of calibrating a crystal to a specified load capacitance.

By subtracting the resonant frequency of the crystal from its antiresonant frequency, one can calculate the range of frequencies over which the crystal reactance is positive:

$$f_a - f_s = f_s \left(\sqrt{1 + C_1/C_0} - 1 \right)$$

$$f_s \left(\frac{C_1}{2C_0} \right)$$

Given typical values for C_1 and C_0 , this range can hardly exceed 0.5% of f_s . Unless the inverting amplifier in the positive reactance oscillator is doing something very strange indeed, the oscillation frequency is bound to be accurate to that percentage whether the crystal was calibrated for series operation or to any unspecified load capacitance.

Equivalent Series Resistance

ESR is the real part of Z_{XTAL} at the oscillation frequency. The oscillation frequency is the parallel resonant frequency of the "XTAL||CL" configuration (which is the same as the series resonant frequency of the "XTAL + CL" configuration). Substituting this frequency into the Z_{XTAL} expression yields, after some algebraic manipulation,

$$ESR = \frac{R_1 \left(\frac{C_0 + C_L}{C_L} \right)^2}{1 + \omega^2 C_1^2 \left(\frac{C_0 + C_L}{C_L} \right)^2}$$

$$\approx R_1 \left(1 + \frac{C_0}{C_L} \right)^2$$

Drive Level

The power dissipated by the crystal is $I_1^2 R_1$, where I_1 is the RMS current in the motional arm of the crystal. This current is given by $V_x/|Z_1|$, where V_x is the RMS voltage across the crystal, and $|Z_1|$ is the magnitude of the impedance of the motional arm. At the oscillation frequency, the motional arm is a positive (inductive) reactance in parallel resonance with $(C_0 + C_L)$. Therefore $|Z_1|$ is approximately equal to the magnitude of the reactance of $(C_0 + C_L)$:

$$|Z_1| = \frac{1}{2\pi f(C_0 + C_L)}$$

where f is the oscillation frequency. Then,

$$P = I_1^2 R_1 = \left(\frac{V_x}{|Z_1|} \right)^2 R_1$$

The impedance of the crystal is

The waveform of the voltage across the crystal (XTAL1 to XTAL2) is approximately sinusoidal. If its peak value is V_{CC} , then V_x is $V_{CC}/\sqrt{2}$. Therefore,

$$P = 2R_1 [\pi f(C_0 + C_L) V_{CC}]^2$$

APPENDIX B OSCILLATOR ANALYSIS PROGRAM

The program is written in BASIC. BASIC is excruciatingly slow, but it has some advantages. For one thing, more people know BASIC than FORTRAN. In addition, a BASIC program is easy to develop, modify, and "fiddle around" with. Another important advantage is that a BASIC program can run on practically any small computer system.

Its slowness is a problem, however. For example, the routine which calculates the "start-up time constant" discussed in the text may take several hours to complete. A person who finds this program useful may prefer to convert it to FORTRAN, if the facilities are available.

Limitations of the Program

The program was developed with specific reference to 8051-type oscillator circuitry. That means the on-chip amplifier is a simple inverter, and not a Schmitt Trigger. The 8096, the 80C51, the 80C48 and 80C49 all have simple inverters. The 8096 oscillator is almost identical to the 8051, differing mainly in the input protection circuitry. The CMOS amplifiers have somewhat different parameters (higher gain, for example), and different transition levels than the 8051.

The MCS-48 family is specifically included in the program only to the extent that the input-output curve used in the steady-state analysis is that of a Schmitt Trigger, if the user identifies the device under analysis as an MCS-48 device. The analysis does not include the voltage dependent phase shift of the Schmitt Trigger.

The clamping action of the input protection circuitry is important in determining the steady-state amplitudes. The steady-state routine accounts for it by setting the negative peak of the XTAL1 signal at a level which depends on the amplitude of the XTAL1 signal in accordance with experimental observations. It's an exercise in curve-fitting. A user may find a different type of curve works better. Later steppings of the chips may behave differently in this respect, having somewhat different types of input protection circuitry.

It should be noted that the analysis ignores a number of important items, such as high-frequency effects in the on-chip circuitry. These effects are difficult to predict, and are no doubt dependent on temperature, frequency, and device sample. However, they can be simulated to a reasonable degree by adding an "output capacitance" of about 20 pF to the circuit model (i.e., in parallel with CX2) as described below.

Notes on Using the Program

The program asks the user to input values for various circuit parameters. First the crystal (or ceramic resonator) parameters are asked for. These are R1, L1, C1, and C0. The manufacturer can supply these values for selected samples. To obtain any kind of correlation between calculation and experiment, the values of these parameters must be known for the specific sample in the test circuit. The value that should be entered for C0 is the C0 of the crystal itself plus an estimated 7 pF to account for the XTAL1-to-XTAL2 pin capacitance, plus any other stray capacitance paralleling the crystal that the user may feel is significant enough to be included.

Then the program asks for the values of the XTAL1-to-ground and XTAL2-to-ground capacitances. For CXTAL1, enter the value of the externally connected bulk capacitor plus an estimated 7 pF for pin capacitance. For CXTAL2, enter the value of the externally connected bulk capacitor plus an estimated 7 pF for pin capacitance plus about 20 pF to simulate high-frequency roll-off and phase shifts in the on-chip circuitry.

Next the program asks for values for the small-signal parameters of the on-chip amplifier. Typically, for the 8051/8751,

Amplifier Gain Magnitude = 15
Feedback Resistance = 2300 K Ω
Output Resistance = 2 K Ω

The same values can be used for MCS-48 (NMOS and HMOS) devices, but they are difficult to verify, because the Schmitt Trigger does not lend itself to small-signal measurements.

100 DEFDBL C,D,F,G,L,P,R,S,X

APRIL 8, 1983

200 REM

300 REM *****

400 REM

500 REM

FUNCTIONS

600 REM

700 REM

800 REM FNZM(R,X) = MAGNITUDE OF A COMPLEX NUMBER, !R+JX!

900 DEF FNZM(R,X) = SQR(R^2+X^2)

1000 REM

1100 REM FNZP(R,X) = ANGLE OF A COMPLEX NUMBER

1200 REM = 180/PI*ARCTAN(X/R) IF R<0

1300 REM = 180/PI*ARCTAN(X/R) + 180 IF R<0 AND X<0

1400 REM = 180/PI*ARCTAN(X/R) - 180 IF R<0 AND X>0

1500 DEF FNZP(R,X) = 180/PI*ATN(X/R) - ((SGN(R)-1)*SGN(X)*90)

1600 REM

1700 REM INDUCTIVE IMPEDANCE AT COMPLEX FREQUENCY S+JF (HZ)

1800 REM Z = 2*PI*S*L + J*2*PI*F*L

1900 REM = FNRL(S,L) + JFNXL(F,L)

2000 DEF FNRL(S,L) = 2*PI*S*L

2100 DEF FNXL(F,L) = 2*PI*F*L

2200 REM

2300 REM CAPACITIVE IMPEDANCE AT COMPLEX FREQUENCY S+JF (HZ)

2400 REM Z = 1/[2*PI*(S+JF)*C]

2500 REM = S/[2*PI*(S^2+F^2)*C] + J*(-F)/[2*PI*(S^2+F^2)*C]

2600 REM = FNRC(S,F,C) + JFNXC(S,F,C)

2700 DEF FNRC(S,F,C) = SC/(2*PI*(S^2+F^2)*CC)

2800 DEF JFNXC(S,F,C) = -FC/(2*PI*(S^2+F^2)*CC)

2900 REM

3000 REM RATIO OF TWO COMPLEX NUMBERS

3100 REM RA+JXA = RA*RB+XA*XB XA=RB-RA*XB

3200 REM ----- + J -----

3300 REM RB+JXB = RB^2+XB^2 RB^2+XB^2

3400 REM = FNRR(RA, XA, RB, XB) + JFNXR(RA, XA, RB, XB)

3500 DEF FNRR(RA, XA, RB, XB) = (RA*RB+XA*XB)/(RB^2+XB^2)

3600 DEF JFNXR(RA, XA, RB, XB) = (XA*RB-XB*RA)/(RB^2+XB^2)

3700 REM

3800 REM PRODUCT OF TWO COMPLEX NUMBERS

3900 REM (RA+JXA)*(RB+JXB) = RA*RB-XA*XB + J(XA*RB+RA*XB)

4000 REM = FNRM(RA, XA, RB, XB) + JFNXM(RA, XA, RB, XB)

4100 DEF FNRM(RA, XA, RB, XB) = RA*RB - XA*XB

4200 DEF JFNXM(RA, XA, RB, XB) = XA*RB + RA*XB

4300 REM

4400 REM PARALLEL IMPEDANCES

4500 REM (RA+JXA):(RB+JXB) = (RA+JXA)*(RB+JXB)

4600 REM RA+RB + J(XA+XB)

4700 REM (RA+JXA):(RB+JXB) =

4800 REM RA*(RB^2+XB^2)+RB*(RA^2+XA^2) XA*(RB^2+XB^2)+XB*(RA^2+XA^2)

4900 REM = ----- + J -----

5000 REM (RA+RB)^2 + (XA+XB)^2 (RA+RB)^2 + (XA+XB)^2

5100 REM = FNRP(RA, XA, RB, XB) + JFNXP(RA, XA, RB, XB)

5200 DEF FNRP(RA, XA, RB, XB) = (RA*(RB^2+XB^2) + RB*(RA^2+XA^2))/(RA+RB)^2 + (XA+XB)^2

5300 DEF JFNXP(RA, XA, RB, XB) = (XA*(RB^2+XB^2) + XB*(RA^2+XA^2))/(RA+RB)^2 + (XA+XB)^2

5400 REM

5500 REM *****

5600 REM

5700 REM BEGIN COMPUTATIONS

5800 REM

5900 REM LET PI = 3.141592654#

6000 REM

6100 REM

6200 REM DEFINE CIRCUIT PARAMETERS

6300 REM

6400 REM GOSUB 14500

6500 REM

6600 REM ESTABLISH NOMINAL RESONANT AND ANTIRESONANT CRYSTAL FREQUENCIES

6700 FS = FIX(1/(2*PI*SQR(L1*C1)))

6800 FA = FIX(1/(2*PI*SQR(L1*C1*CO/(C1+CO))))

6900 PRINT

7000 PRINT "XTAL IS SERIES RESONANT AT ",FS," HZ"

7100 PRINT " PARALLEL RESONANT AT ",FA," HZ"

7200 PRINT

7300 PRINT "SELECT: 1 LIST PARAMETERS"

7400 PRINT " 2 CIRCUIT ANALYSIS"

7500 PRINT " 3 OSCILLATION FREQUENCY"

7600 PRINT " 4 START-UP TIME CONSTANT"

7700 PRINT " 5 STEADY-STATE ANALYSIS"

230659-44


```

7900 PRINT
8000 INPUT N
8100 IF N=1 THEN PRINT ELSE 8600
8200 REM
8300 REM ----- LIST PARAMETERS -----
8400 GOSUB 17100
8500 GOTO 6800
8600 IF N=2 THEN PRINT ELSE 9400
8700 REM
8800 REM ----- CIRCUIT ANALYSIS -----
8900 PRINT " FREQUENCY S+JF TYPE (S), (F) "
9000 INPUT SQ,FQ
9100 GOSUB 20200
9200 GOSUB 26600
9300 GOTO 6800
9400 IF N=3 THEN 10300 ELSE 11000
9500 REM
9600 REM ----- OSCILLATION FREQUENCY -----
9700 CL = CX*CY/(CX+CY) + CO
9800 FQ = FIX(1/(2*PI*SQR(L1*C1*CL/(C1+CL))))
9900 SQ = 0
10000 DF = FIX(10*INT(LOG(FA-FS)/LOG(10)-2)+.5)
10100 DS = 0
10200 RETURN
10300 GOSUB 9700
10400 GOSUB 30300
10500 PRINT
10600 PRINT
10700 PRINT "FREQUENCY AT WHICH LOOP GAIN HAS ZERO PHASE ANGLE. "
10800 GOSUB 26600
10900 GOTO 6800
11000 IF N=4 THEN PRINT ELSE 12200
11100 REM
11200 REM ----- START-UP TIME CONSTANT -----
11300 PRINT "THIS WILL TAKE SOME TIME. "
11400 GOSUB 9700
11500 GOSUB 37700
11600 PRINT
11700 PRINT
11800 PRINT "FREQUENCY AT WHICH LOOP GAIN = 1 AT 0 DEGREES. "
11900 GOSUB 26600
12000 PRINT "PRINT "THIS YIELDS A START-UP TIME CONSTANT OF ",CSNG(1000000/(2*PI*SQ))," MICROSECS"
12100 GOTO 6800
12200 IF N=5 THEN PRINT ELSE 7300
12300 REM
12400 REM ----- STEADY-STATE ANALYSIS -----
12500 PRINT "STEADY-STATE ANALYSIS"
12600 PRINT
12700 PRINT "SELECT:  1. 8031/8051"
12800 PRINT "          2. 8751"
12900 PRINT "          3. 8035/8039/8040/8048/8049"
13000 PRINT "          4. 8748/8749"
13100 INPUT ICX
13200 IF ICX<1 OR ICX>4 THEN 12600
13300 GOSUB 46900
13400 GOTO 7300
13500 REM SUBROUTINE BELOW DEFINES INPUT-OUTPUT CURVE OF OSCILLATOR CKT
13600 IF ICX>2 AND VO=5 AND VI<2 THEN RETURN
13700 VO = -10*VI + 15
13800 IF VO>5 THEN VO = 5
13900 IF VO<2 THEN VO = 2
14000 IF ICX>2 AND VO>2 THEN VO = 5
14100 RETURN
14200 REM
14300 REM *****
14400 REM
14500 REM DEFINE CIRCUIT PARAMETERS
14600 REM
14700 INPUT " R1 (OHMS):",R1
14800 INPUT " L1 (HENRY):",L1
14900 INPUT " C1 (PF):",X
15000 C1 = X*1E-12
15100 INPUT " C0 (PF):",X
15200 C0 = X*1E-12
15300 INPUT " CXTAL1 (PF):",X
15400 CX = X*1E-12
15500 INPUT " CXTAL2 (PF):",X
15600 CY = X*1E-12

```

230659-45

```

15700 INPUT " GAIN FACTOR MAGNITUDE",AV#
15800 INPUT " AMP FEEDBACK RESISTANCE (K-OHMS)",X
15900 RX = X*1000#
16000 INPUT " AMP OUTPUT RESISTANCE (K-OHMS)",X
16100 RO = X*1000#
16200 REM
16300 REM
16400 REM          LIST CURRENT PARAMETER VALUES
16500 GOSUB 17100
16600 RETURN
16700 REM
16800 REM
16900 REM *****
17000 REM
17100 REM          LIST CURRENT PARAMETER VALUES
17200 REM
17300 PRINT
17400 PRINT "CURRENT PARAMETER VALUES  1. R1 = ",R1," OHMS"
17500 PRINT "                               2. L1 = ",CSNG(L1)," HENRY"
17600 PRINT "                               3. C1 = ",CSNG(C1*1E+12)," PF"
17700 PRINT "                               4. CO = ",CSNG(CO*1E+12)," PF"
17800 PRINT "                               5. CXTAL1 = ",CSNG(CX*1E+12)," PF"
17900 PRINT "                               6. CXTAL2 = ",CSNG(CY*1E+12)," PF"
18000 PRINT "       7. AMPLIFIER GAIN MAGNITUDE = ",AV#
18100 PRINT "       8. FEEDBACK RESISTANCE = ",CSNG(RX* 001)," K-OHMS"
18200 PRINT "       9. OUTPUT RESISTANCE = ",CSNG(RO* 001)," K-OHMS"
18300 PRINT
18400 PRINT "TO CHANGE A PARAMETER VALUE, TYPE (PARAM NO ), (NEW VALUE)."
18500 PRINT "OTHERWISE, TYPE 0,0"
18600 INPUT NX,X
18700 IF NX=0 THEN RETURN
18800 IF NX=1 THEN R1 = X
18900 IF NX=2 THEN L1 = X
19000 IF NX=3 THEN C1 = X*1E-12
19100 IF NX=4 THEN CO = X*1E-12
19200 IF NX=5 THEN CX = X*1E-12
19300 IF NX=6 THEN CY = X*1E-12
19400 IF NX=7 THEN AV# = X
19500 IF NX=8 THEN RX = X*1000#
19600 IF NX=9 THEN RO = X*1000#
19700 GOTO 17400
19800 REM
19900 REM
20000 REM *****
20100 REM
20200 REM          CIRCUIT ANALYSIS
20300 REM
20400 REM This routine calculates the loop gain at complex frequency SQ+jFQ.
20500 REM
20600 REM 1. Crystal impedance RE + jXE
20700 REM
20800 X1 = FNXL(FQ,L1) + FNXC(SQ,FQ,C1)
20900 RE = FNRP((R1+FNRL(SQ,L1)+FNRC(SQ,FQ,C1)),X1,FNRC(SQ,FQ,CO),FNXC(SQ,FQ,CO))
21000 XE = FNXP((R1+FNRL(SQ,L1)+FNRC(SQ,FQ,C1)),X1,FNRC(SQ,FQ,CO),FNXC(SQ,FQ,CO))
21100 REM
21200 REM 2. RF + jXF = (RE+jXE):(amplifier feedback resistance)
21300 REM
21400 RF = FNRP(RX,O,RE,XE)
21500 XF = FNXP(RX,O,RE,XE)
21600 REM
21700 REM 3. Input impedance, Zi = RI + jXI = impedance of CXTAL1
21800 REM
21900 RI = FNRC(SQ,FQ,CX)
22000 XI = FNXC(SQ,FQ,CX)
22100 REM
22200 REM 4. Load impedance: ZL = (impedance of CXTAL2):[(RF+RI)+j(XF+XI)]
22300 REM
22400 RL = FNRP((RF+RI),(XF+XI),FNRC(SQ,FQ,CY),FNXC(SQ,FQ,CY))
22500 XL = FNXP((RF+RI),(XF+XI),FNRC(SQ,FQ,CY),FNXC(SQ,FQ,CY))
22600 REM
22700 REM 5. Amplifier gain A = -AV*ZL/(ZL+RO)
22800 REM          = A:real) + jA:imaginary)
22900 REM
23000 AR# = -AV#*FNRR(RL,XL,(RO+RL),XL)
23100 AI# = -AV#*FNXR(RL,XL,(RO+RL),XL)
23200 REM
23300 REM 6. Feedback ratio (beta) = (RI+jXI)/[(RF+RI)+j(XF+XI)]
23400 REM          = B:real) + jB:imaginary)

```

```

23500 REM
23600 BR# = FNRR(RI, XI, (RI+RF), (XI+XF))
23700 BI# = FNXR(RI, XI, (RI+RF), (XI+XF))
23800 REM
23900 REM 7. Amplifier gain in magnitude/phase form AR+jAI = A at AP degrees
24000 REM
24100 A = FNZM(AR#, AI#)
24200 AP = FNZP(AR#, AI#)
24300 REM
24400 REM 8 (beta) in magnitude/phase form BR+jBI = B at BP degrees
24500 REM
24600 B = FNZM(BR#, BI#)
24700 BP = FNZP(BR#, BI#)
24800 REM
24900 REM 9 Loop gain G = (BR+jBI)*(AR+jAI)
25000 REM
25100 REM
25200 GR = FNRM(AR#, AI#, BR#, BI#)
25300 GI = FNXM(AR#, AI#, BR#, BI#)
25400 REM
25500 REM 10. Loop gain in magnitude/phase form. GR+jGI = AL at AQ degrees
25600 REM
25700 AL = FNZM(GR, GI)
25800 AQ = FNZP(GR, GI)
25900 RETURN
26000 REM
26100 REM
26200 REM *****
26300 REM
26400 REM PRINT CIRCUIT ANALYSIS RESULTS
26500 REM
26600 PRINT
26700 PRINT " FREQUENCY = ", SQ, " + j", FQ, " HZ"
26800 PRINT " XTAL IMPEDANCE = ", FNZM(RE, XE), " OHMS AT ", FNZP(RE, XE), " DEGREES"
26900 PRINT " (RE = ", CSNG(RE), " OHMS)"
27000 PRINT " (XE = ", CSNG(XE), " OHMS)"
27100 PRINT " LOAD IMPEDANCE = ", FNZM(RL, XL), " OHMS AT ", FNZP(RL, XL), " DEGREES"
27200 PRINT " AMPLIFIER GAIN = ", A, " AT ", AP, " DEGREES"
27300 PRINT " FEEDBACK RATIO = ", B, " AT ", BP, " DEGREES"
27400 PRINT " LOOP GAIN = ", AL, " AT ", AQ, " DEGREES"
27500 RETURN
27600 REM
27700 REM
27800 REM *****
27900 REM
28000 REM SEARCH FOR FREQUENCY (S+JF)
28100 REM AT WHICH LOOP GAIN HAS ZERO PHASE ANGLE
28200 REM
28300 REM This routine searches for the frequency at which the imaginary part
28400 REM of the loop gain is zero. The algorithm is as follows:
28500 REM 1. Calculate the sign of the imaginary part of the loop gain (GI).
28600 REM 2. Increment the frequency.
28700 REM 3. Calculate the sign of GI at the incremented frequency.
28800 REM 4. If the sign of GI has not changed, go back to 2.
28900 REM 5. If the sign of GI has changed, and this frequency is within
29000 REM 1Hz of the previous sign-change, exit the routine.
29100 REM 6. Otherwise, divide the frequency increment by -10.
29200 REM 7. Go back to 2.
29300 REM The routine is entered with the starting frequency SQ+jFQ and
29400 REM starting increment DS+jDF already defined by the calling program.
29500 REM In actual use either DS or DF is zero, so the routine searches for
29600 REM a GI=0 point by incrementing either SQ or FQ while holding the other
29700 REM constant. It returns control to the calling program with the
29800 REM incremented part of the frequency being within 1Hz of the actual
29900 REM GI=0 point.
30000 REM
30100 REM 1. CALCULATE THE SIGN OF THE IMAGINARY PART OF THE LOOP GAIN (GI).
30200 REM
30300 GOSUB 20200
30400 GOSUB 26600
30500 IF GI=0 THEN RETURN
30600 SX% = INT(SGN(GI))
30700 IF SX%=-1 THEN DS = -DS
30800 REM (REVERSAL OF DS FOR GI=0 IS FOR THE POLE-SEARCH ROUTINE)
30900 REM
31000 REM 2. INCREMENT THE FREQUENCY.
31100 REM
31200 SP = SQ

```

```

31300 FP = FQ
31400 SQ = SQ + DS
31500 FQ = FQ + DF
31600 REM
31700 REM 3 CALCULATE THE SIGN OF G1 AT THE INCREMENTED FREQUENCY.
31800 REM
31900 GOSUB 20200
32000 GOSUB 26600
32100 IF INT(SGN(G1))=0 THEN RETURN
32200 REM
32300 REM 4 IF THE SIGN OF G1 HAS NOT CHANGED, GO BACK TO 2.
32400 REM
32500 IF SX+INT(SGN(G1))=0 THEN PRINT ELSE 31400
32600 SX = -SX
32700 REM
32800 REM 5 IF THE SIGN OF G1 HAS CHANGED, AND IF THIS FREQUENCY IS WITHIN
32900 REM 1HZ OF THE PREVIOUS SIGN-CHANGE, AND IF G1 IS NEGATIVE, THEN
33000 REM EXIT THE ROUTINE (THE ADDITIONAL REQUIREMENT FOR NEGATIVE G1
33100 REM IS FOR THE POLE-SEARCH ROUTINE)
33200 REM
33300 IF ABS(SP-SQ)<1 AND ABS(FP-FQ)<1 AND SX=-1 THEN RETURN
33400 REM
33500 REM 6. DIVIDE THE FREQUENCY INCREMENT BY -10
33600 REM
33700 DS = -DS/10#
33800 DF = -DF/10#
33900 REM
34000 REM 7. GO BACK TO 2
34100 REM
34200 GOTO 31200
34300 REM
34400 REM
34500 REM *****
34600 REM
34700 REM SEARCH FOR POLE FREQUENCY
34800 REM
34900 REM This routine searches for the frequency at which the loop gain = 1
35000 REM at 0 degrees. That frequency is the pole frequency of the closed-
35100 REM loop gain function. The pole frequency is a complex number, SQ+jFQ
35200 REM (Hz). Oscillator start-up ensues if SQ>0. The algorithm is based on
35300 REM the calculated behavior of the phase angle of the loop gain in the
35400 REM region of interest on the complex plane. The locus of points of zero
35500 REM phase angle crosses the j-axis at the oscillation frequency and at
35600 REM some higher frequency. In between these two crossings of the j-axis,
35700 REM the locus lies in Quadrant I of the complex plane, forming an
35800 REM approximate parabola which opens to the left. The basic plan is to
35900 REM follow the locus from where it crosses the j-axis at the oscillation
36000 REM frequency, into Quadrant I, and find the point on that locus where
36100 REM the loop gain has a magnitude of 1. The algorithm is as follows:
36200 REM 1. Find the oscillation frequency, 0+jFQ
36300 REM 2. At this frequency calculate the sign of (AL-1). (AL = magnitude
36400 REM of loop gain.)
36500 REM 3. Increment FQ
36600 REM 4. For this value of FQ, find the value of SQ for which the loop
36700 REM gain has zero phase.
36800 REM 5. For this value of SQ+jFQ, calculate the sign of (AL-1).
36900 REM 6. If the sign of (AL-1) has not changed, go back to 3.
37000 REM 7. If the sign of (AL-1) has changed, and this value of FQ is
37100 REM within 1Hz of the previous sign-change, exit the routine.
37200 REM 8. Otherwise, divide the FQ-increment by -10.
37300 REM 9. Go back to 3.
37400 REM
37500 REM 1. FIND THE OSCILLATION FREQUENCY, 0+jFQ
37600 REM
37700 GOSUB 9700
37800 GOSUB 30300
37900 REM
38000 REM 2. AT THIS FREQUENCY, CALCULATE THE SIGN OF (AL-1).
38100 REM
38200 SYX = INT(SGN(AL-1))
38300 IF SYX=-1 THEN STOP
38400 REM ESTABLISH INITIAL INCREMENTATION VALUE FOR FQ
38500 F1 = FQ
38600 DF = (FA-F1)/10#
38700 GOSUB 30300
38800 DE = (FG-F1)/10#
38900 DF = 0
39000 FG = F1

```

TA-000005

230659-48


```

39100 REM
39200 REM 3. INCREMENT FQ
39300 REM
39400 FQ = FQ + DE
39500 REM
39600 REM 4. FOR THIS VALUE OF FQ, FIND THE VALUE OF SQ FOR WHICH THE LOOP
39700 REM GAIN HAS ZERO PHASE. (THE ROUTINE WHICH DOES THAT NEEDS DF = 0,
39800 REM SO THAT IT CAN HOLD FQ CONSTANT, AND NEEDS AN INITIAL VALUE FOR
39900 REM DS, WHICH IS ARBITRARILY SET TO DS = 1000.)
40000 REM
40100 DS = 1000#
40200 SQ = 0
40300 GOSUB 30300
40400 IF AL=1 THEN RETURN
40500 REM
40600 REM 5. FOR THIS VALUE OF SQ+JFQ, CALCULATE THE SIGN OF (AL-1).
40700 REM 6. IF THE SIGN OF (AL-1) HAS NOT CHANGED, GO BACK TO 3.
40800 REM
40900 IF SYX+INT(SGN(AL-1))=0 THEN PRINT ELSE 39400
41000 REM
41100 REM 7. IF THE SIGN OF (AL-1) HAS CHANGED, AND THIS VALUE OF FQ IS WITHIN
41200 REM 1HZ OF THE PREVIOUS SIGN-CHANGE, EXIT THE ROUTINE.
41300 REM
41400 IF ABS(F1-FQ)<1 THEN RETURN
41500 REM
41600 REM 8. DIVIDE THE FQ-INCREMENT BY -10.
41700 REM
41800 DE = -DE/10#
41900 F1 = FQ
42000 SYX = -SYX
42100 REM
42200 REM 9. GO BACK TO 3.
42300 REM
42400 GOTO 39400
42500 REM
42600 REM
42700 REM *****
42800 REM
42900 REM
43000 REM
43100 REM The circuit model used in this analysis is similar to the one used
43200 REM in the small-signal analysis, but differs from it in two respects.
43300 REM First, it includes clamping and clipping effects described in the
43400 REM text. Second, the voltage source in the Thevenin equivalent of the
43500 REM amplifier is controlled by the input voltage in accordance with an
43600 REM input-output curve defined elsewhere in the program.
43700 REM The analysis applies a sinusoidal input signal of arbitrary
43800 REM amplitude, at the oscillation frequency, to the XTAL1 pin, then
43900 REM calculates the resulting waveform from the voltage source. Using
44000 REM standard Fourier techniques, the fundamental frequency component of
44100 REM this waveform is extracted. This frequency component is then
44200 REM multiplied by the factor  $1/ZL/(ZL+RO)$ , and the result is taken to be
44300 REM the signal appearing at the XTAL2 pin. This signal is then
44400 REM multiplied by the feedback ratio (beta), and the result is taken to
44500 REM be the signal appearing at the XTAL1 pin. The algorithm is now
44600 REM repeated using this computed XTAL1 signal as the assumed input
44700 REM sinusoid. Every time the algorithm is repeated, new values appear at
44800 REM XTAL1 and XTAL2, but the values change less and less with each
44900 REM repetition. Eventually they stop changing. This is the steady-state.
45000 REM The algorithm is as follows:
45100 REM 1. Compute approximate oscillation frequency.
45200 REM 2. Call a circuit analysis at this frequency.
45300 REM 3. Find the quiescent levels at XTAL1 and XTAL2 (to establish the
45400 REM beginning DC level at XTAL1).
45500 REM 4. Assume an initial amplitude for the XTAL1 signal.
45600 REM 5. Correct the DC level at XTAL1 for clamping effects, if necessary.
45700 REM 6. Using the appropriate input-output curve, extract a DC level and
45800 REM the fundamental frequency component (multiplying the latter by
45900 REM  $1/ZL/(ZL+RO)$ ).
46000 REM 7. Clip off the negative portion of this output signal, if the
46100 REM negative peak falls below zero.
46200 REM 8. If this signal, multiplied by (beta), differs from the input
46300 REM amplitude by less than 1mV, or if the algorithm has been repeated
46400 REM 10 times, exit the routine.
46500 REM 9. Otherwise, multiply the XTAL2 amplitude by (beta) and feed it
46600 REM back to XTAL1, and go back to 5.
46700 REM
46800 REM 1. COMPUTE APPROXIMATE OSCILLATION FREQUENCY.

```

```

46900 GOSUB 9700
47000 REM
47100 REM 2. CALL A CIRCUIT ANALYSIS AT THIS FREQUENCY.
47200 GOSUB 20800
47300 PRINT PRINT : PRINT "ASSUMED OSCILLATION FREQUENCY:"
47400 GOSUB 26600
47500 PRINT : PRINT
47600 REM
47700 REM 3. FIND QUIESCENT POINT
47800 REM (At quiescence the voltages at XTAL1 and XTAL2 are equal. This
47900 REM voltage level is found by trial-and-error, based on the input-
48000 REM output curve, so that a person can change the input-output curve
48100 REM as desired without having to re-calculate the quiescent point.
48200 VI = 0
48300 VB = 1
48400 K1 = 1
48500 VI = VI + VB
48600 GOSUB 13600
48700 IF ABS(VD-VI)<.001 THEN 49200
48800 IF K1+SGN(VD-VI)=0 THEN 48900 ELSE 48500
48900 K1 = SGN(VD-VI)
49000 VB = -VB/10
49100 GOTO 48500
49200 VB = VI
49300 PRINT "QUIESCENT POINT = ",VB
49400 REM
49500 REM 4. ASSUME AN INITIAL AMPLITUDE FOR THE XTAL1 SIGNAL.
49600 EI = .01
49700 NRX = 0
49800 REM
49900 REM 5. CORRECT FOR CLAMPING EFFECTS, IF NECESSARY.
50000 REM (K1 and K2 are curve-fitting parameters for the ROM parts.)
50100 K1 = (2.5-VB)/(3-VB)
50200 K2 = (VB-1.25)/(3-VB)
50300 IF ICX=2 OR ICX=4 THEN IF EI<(VB+5) THEN EO = VB ELSE EO = EI - .5
50400 IF ICX=1 OR ICX=3 THEN IF EI<(VB+5) THEN EO = VB ELSE EO = K1*EI+K2
50500 NRX = NRX + 1
50600 REM
50700 REM 6. DERIVE XTAL2 AMPLITUDE
50800 VO = 0
50900 VC = 0
51000 VS = 0
51100 FOR NX = -25 TO +24
51200 VI = EO - EI*COS(PI*NX/25)
51300 GOSUB 13600
51400 VO = VO + VD
51500 VC = VC + VD*COS(PI*NX/25)
51600 VS = VS + VD*SIN(PI*NX/25)
51700 NEXT NX
51800 VO = VO/50
51900 V1 = SGR(VC^2+VS^2)/25*FNZM(RL,XL)/FNZM((RL+RO),XL)
52000 REM
52100 REM 7. CLIP XTAL2 SIGNAL
52200 IF VO-V1<0 THEN VL = 0 ELSE VL = VO-V1
52300 PRINT PRINT : PRINT "XTAL1 SWING = ",EO-EI," TO ",EO+EI
52400 PRINT "XTAL2 SWING = ",VL," TO ",VO+V1
52500 REM
52600 REM 8. TEST FOR TERMINATION
52700 IF ABS(EI-V1*B)<.001 OR NRX=10 THEN RETURN
52800 REM
52900 REM 9. FEED BACK TO XTAL1 AND REPEAT
53000 EI = V1*B
53100 GOTO 50300

```

230659-50

1 COMPUTE APPROXIMATE OSCILLATION FREQUENCY

2 Assume an initial amplitude for the XTAL1 signal.

3 Beginning DC level at XTAL1.

4 Find the quiescent levels at XTAL1 and XTAL2.

5 Call a circuit analysis at this frequency.

6 Compute approximate oscillation.

7 The algorithm is as follows:

8 a. Assume an initial amplitude for the XTAL1 signal.

9 b. Compute the DC level at XTAL1 for clamping effects. If necessary

10 Using the approximate input-output curve, extract a DC level and

11 the fundamental frequency component (multiplying the latter by

12 $1/(1+R/L)$).

13 c. Compute the negative portion of the output signal. If the

14 negative part falls below zero

15 d. This signal, multiplied by factor, differs from the input

16 amplitude by less than 10% or if the algorithm has been repeated

17 10 times, exit the routine.

18 Otherwise, multiply the XTAL2 amplitude by (data) and feed it

19 back to XTAL1, and go back to 5.

INTRODUCTION

What's the driving factor in your embedded control application? Board space? Reliability? Power? Design time? Manufacturing simplicity? Cost?

What if a single component helped you achieve smaller board size, higher reliability, lower power, faster design time, simplified manufacturing, and lower cost? Intel's 87C75PF is the first in a family of microcontroller peripheral port expander products. This application note will show how the 87C75PF significantly reduces chip count and greatly simplifies system design. The 87C75PF data sheet has detailed device information.

Intel's early microcontrollers had obvious benefits over previous alternatives — a high degree of system integration. The most common microcomputer functions — CPU, ROM, RAM, I/O ports, timers/counters, address decoding, etc. — were combined onto a single chip. Upgrades and proliferations have grown significantly since those early days. Four-bit and 8-bit controllers are the most widely used, with 16-bit versions, spearheaded by Intel's 8096 family, beginning their exponential growth.

The most sought after microcontroller improvement is additional program memory. 8- and 16-bit controllers are optionally equipped with 4K or 8K bytes of ROM or EPROM. This is sufficient memory for about half of embedded applications.

The remaining applications use off-chip EPROM. One reason, of course, is to increase system memory; typically to 16K- or 32K-bytes. Another is to provide flexibility for code that changes frequently. In other applications, generic boards or multi-use modules can be manufactured and custom-programmed for special con-

figurations. For example, a single robot control module can be manufactured. Identical robots can be configured to perform various factory tasks.

8- and 16-bit microcontrollers accommodate external-memory expansion. Controllers sacrifice two 8-bit I/O ports to supply address and data lines to peripheral components. Unfortunately, expanded-memory modes violate two embedded-control objectives: maximizing I/O capability and reducing chip count (or board size). Usually, systems that need more memory are also I/O intensive. Traditional memory-expansion/port-recovery schemes use multiple chips. Memory, address latches, port latches, transceivers, address decoders, and glue chips turn a single-chip uC system into a multiple-chip conglomeration.

THE MULTIPLEXED BUS

To achieve small board size, embedded control systems require minimum chip count and chips that occupy small footprints. Embedded controllers use multiplexed address/data buses to achieve both. An 8051 controller, for example, shares its lower eight address pins with its 8-bit data.

Every memory access requires two cycles — one for address, one for data (see Figure 1). The controller's first cycle places a 16-bit address on the bus. It holds the upper eight bits constant throughout the access. It presents the low-address byte just long enough for an external latch to capture it. The latch and controller's upper bus then supply the 16-bit address to external devices for the remainder of the memory access. The controller's data cycle transmits or receives data on its multiplexed lower address/data pins. The multiplexed bus minimizes the controller's pin count and the system's board traces.

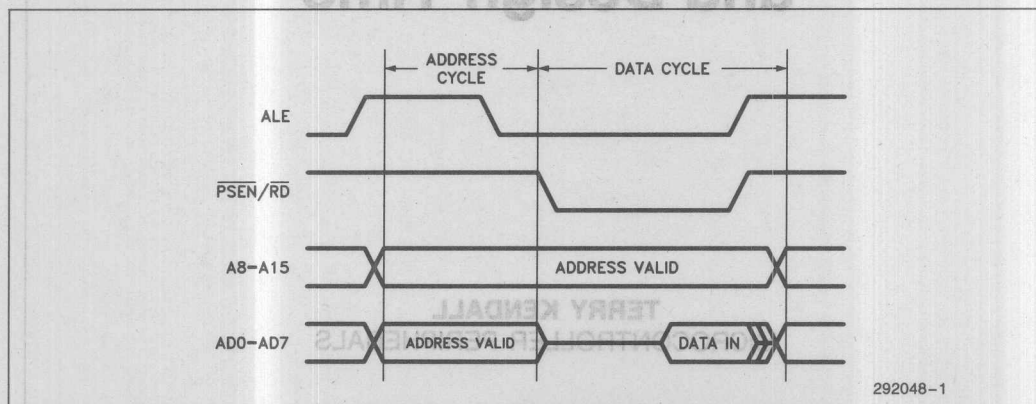


Figure 1. Every microcontroller memory access requires two cycles.

SYSTEM PERFORMANCE

Every system component influences performance. Performance encompasses speed, system noise, and power consumption. A typical expanded-mode controller application uses many chips to increase memory and recover lost I/O. Figure 3 shows an improved, but more expensive, alternative to the system in Figure 2. "Glue" chips between the controller and peripherals delay address signals. To optimize system speed, fast, expensive glue chips, memory, and peripheral devices are required.

Multiple-chip solutions consume significant power and inject noise into a system. A beefed-up, well regulated power supply will relieve symptoms, but adds significantly to cost, board size, and weight.

THE 87C75PF SOLUTION

Figure 4 shows the same system using the 87C75PF — a two chip solution!

The 87C75PF furnishes a no-glue interface to 8051-based systems and all other Intel-architecture embedded controllers. The Port Expander's flexible, user-programmable memory map and alterable control signals simplify 8051, 8096, and 80188 connections.

Examples in this application note show how the 87C75PF works with various microcontrollers. An 8051/87C75PF system that takes advantage of high-level compiled languages and an in-system programmable example will also be shown.

SYSTEM INTEGRATION

Intuitively we all recognize the benefit of system integration — chip-count is reduced.

Just as important are:

- small board size with few layers
- increased performance
- decreased design time
- optimized software development
- reduced inventory
- less incoming inspection
- increased system reliability
- simplified manufacturing.

Cost is a prime consideration. The itemized cost of discrete components is only one parameter. Until the benefits listed above are quantified, realistic system costs can't be determined. Hardware design and software development time are significant up-front expenses. Multiple-chip systems incur substantial inventory, incoming inspection, testing, manufacturing, board size, and rework costs.

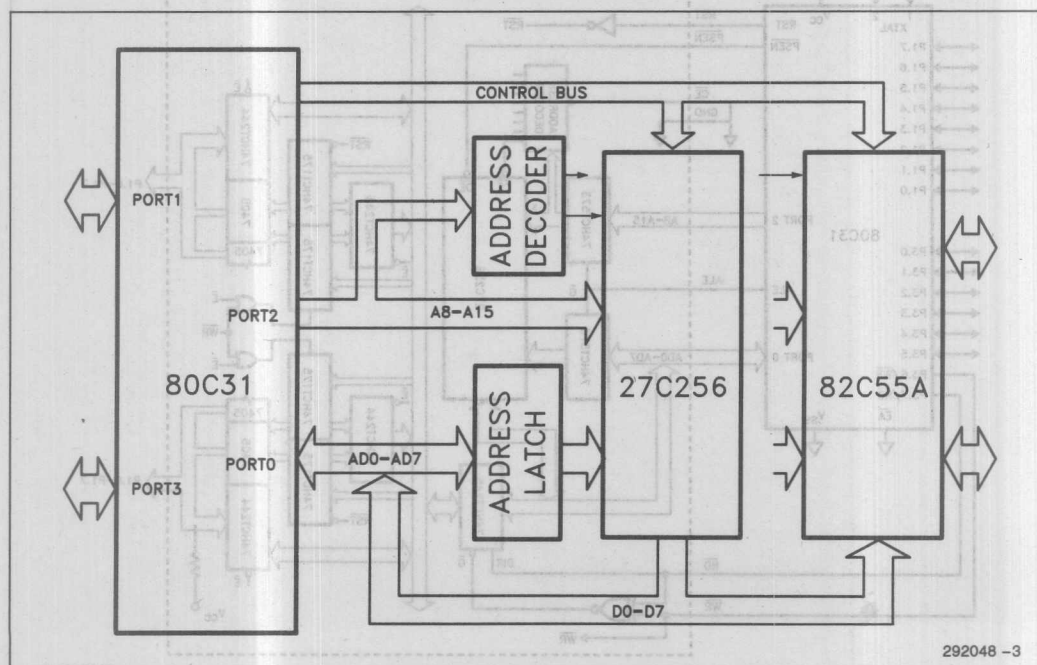


Figure 3. A simplified multiple-chip system.

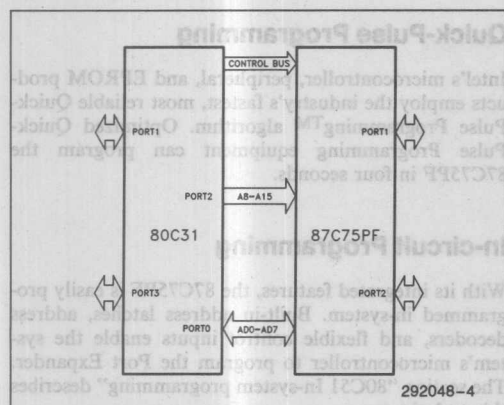


Figure 4. A "no-glue", two chip 87C75PF system.

Reliability also has significant value — to you and your customers. Customers demand products that work properly — forever. Reworked products waste time and money, increase the cost of every unit you ship, and ruin your company's reputation. The best way to increase reliability is to eliminate system components.

Simplified manufacturing saves time and money while increasing reliability. One factory-tested, integrated-function chip is much easier to place on a circuit board and is far more reliable than myriad discrete chips. Every solder joint is a possible failure point. A single chip reduces potential failure points from hundreds to a few.

87C75PF ARCHITECTURE

The 87C75PF Port Expander's features include:

- Two 8-bit I/O ports
- 32K × 8 EPROM
- Two 64K-byte memory planes
- Special Function Registers
- Device-configuration registers
- "No-glue" controller interface
- Low-power, Low-noise CHMOSII-E
- Quick-Pulse Programming™ Algorithm
- In-system programmability
- 40-Pin Cerdip, 44-Lead PLCC packages

Two Ports

The 87C75PF has two 8-bit bi-directional I/O ports. Port 1 has open-drain outputs and port 2 has quasi-bi-directional (resistor pull-up) outputs. Each port is individually addressable with separate port-latch and port-pin addresses. Typical of quasi-bi-directional ports, they are always in output mode but can be used as inputs by simply writing logic "1s" to their latches.

Relocatable EPROM

The EPROM has 262,144 bits organized as 32K 8-bit words. Its access time determines the device's speed rating. The 32K-byte EPROM occupies half of the program memory (or EPROM) plane. The EPROM block can be located in either the lower or upper half of the EPROM plane to accommodate various microcontroller architectures.

Dual or Single Memory Planes

8051-family microcontrollers have two external memory planes — program and data. 8096-, 80188-, and 68xx-family microcontrollers have only one program/data plane. The 87C75PF's user-configurable double- and single-plane modes work with any 8-bit microcontroller architecture.

Relocatable SFRs

The 87C75PF has five special function registers:

- Port 1 latch
- Port 2 latch
- Port 1 pin
- Port 2 pin
- Plane select.

Port-latch registers allow the microcontroller to change port-pin output levels. The microcontroller can read the port latches to recall the last value written. A microcontroller can determine external pin levels by reading the port-pin locations.

During programming, the plane select register determines whether the EPROM array or the configuration registers are being programmed. More special function register details are described later in this application note.

Device Reconfiguration

Non-volatile (EPROM cell) device-configuration registers configure the 87C75PF for microcontroller compatibility. Programmable configuration registers can:

- relocate the EPROM array in the memory map
- relocate the SFRs in the memory map
- combine the EPROM and SFR planes
- change the reset pin's active polarity
- insert transistor pull-ups on port pins.

In its default configuration, the 87C75PF is compatible with the 8051's two-plane architecture. It is easily reconfigured for single-plane 8096 architecture. Remapping the memory planes makes the device compatible with 80188 and 68xx architectures.

Various microcontrollers have different reset input levels. The 8051's reset is active-high while the 8096's is active-low. The 80188 has an active-low reset input and active-high synchronous reset output. The Port Expander's configurable reset polarity can work with active-high or active-low microcontrollers.

If the I/O ports are used only as outputs, a "push-pull" drive is desirable. Port 1 and/or port 2 can be configured to have active pull-up transistors rather than open-drain or quasi-bi-directional outputs.

"No-glue" Microcontroller Interface

The 87C75PF's internal address latches, address decoders, reconfigurable memory planes, and alterable control inputs allow no-glue interfacing to any Intel microcontroller. The 87C75PF makes expanded-mode, two-chip microcontroller systems a reality.

Quick-Pulse Programming

Intel's microcontroller, peripheral, and EPROM products employ the industry's fastest, most reliable Quick-Pulse Programming™ algorithm. Optimized Quick-Pulse Programming equipment can program the 87C75PF in four seconds.

In-circuit Programming

With its integrated features, the 87C75PF is easily programmed in-system. Built-in address latches, address decoders, and flexible control inputs enable the system's microcontroller to program the Port Expander. The section "80C51 In-system programming" describes this technique.

Packaging

For systems requiring periodic reprogramming, prototyping, or hermetic packages, the 87C75PF is available in a 40-pin ceramic DIP (CERDIP) package. PLCC packaging is available to further reduce board size and provide for surface mount and automated manufacturing.

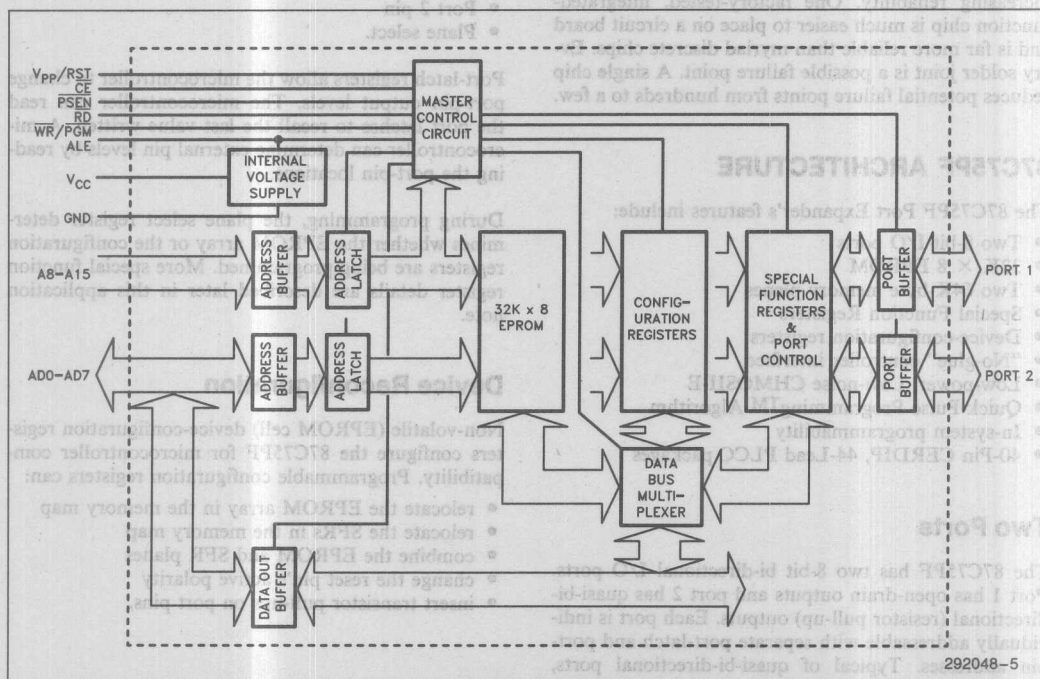


Figure 5. 87C75PF Block Diagram.

87C75PF FUNCTIONAL BLOCKS

Figure 5 shows the 87C75PF's block diagram. The device has three main functional blocks, or memory planes: EPROM, special function registers, and configuration registers.

The block diagram shows device inputs on the left and outputs on the right. Sixteen address lines enter the device and their states are latched by ALE. The lower eight address pins are multiplexed with data. PSEN (Program-Store ENable) gates the device's EPROM data. RD gates SFR data. WR/PGM controls SFR data writes. CE is the master chip enable input. Vpp (the programming voltage input) is multiplexed with RST (reset). Vpp is required only during programming. Asserting RST sets port latches to "1s" during operating mode.

Port 1 is an 8-bit open-drain port with optional "CMOS" drive capability. Port 2 has 8 quasi-bi-directional pins, also with optional "CMOS" drive.

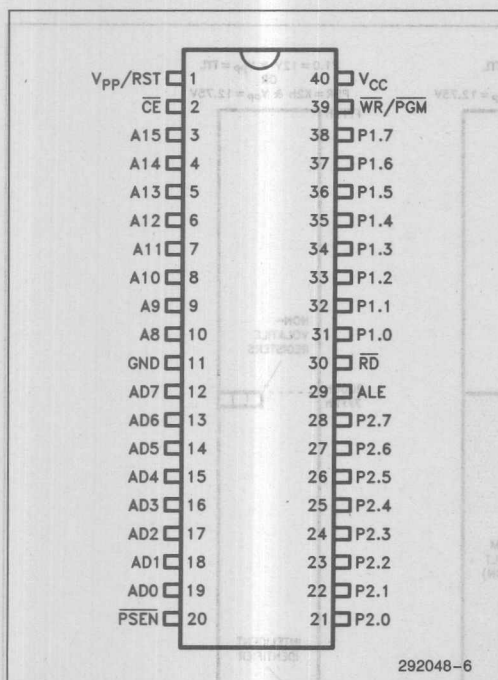


Figure 6. DIP Pinout.

DEVICE PINOUTS

The 87C75PF is available in two package styles — 40-pin Cerdip and 44-lead PLCC. Both pinouts are similar to Intel's 27210 megabit EPROM. The device's pinouts are compatible with most programming equipment capable of programming 27210 EPROMs.

Figure 6 shows the Cerdip pinout. The left side has sequential address and data inputs. The ground pin (GND) separates lower and upper address lines for better noise immunity. Ports are logically placed on the device's right side. Port 1, which is open-drain, is near VCC. SIP-pack resistor pull-ups added externally to port 1 have easy access to VCC.

Figure 7 shows the PLCC pinout. PLCC leads are in the same sequence as the Cerdip pinout. No-connect (NC) and don't-use (DU) leads are inserted at strategic locations. Future enhancements will use these leads for expanded features. DU leads should be left unconnected.

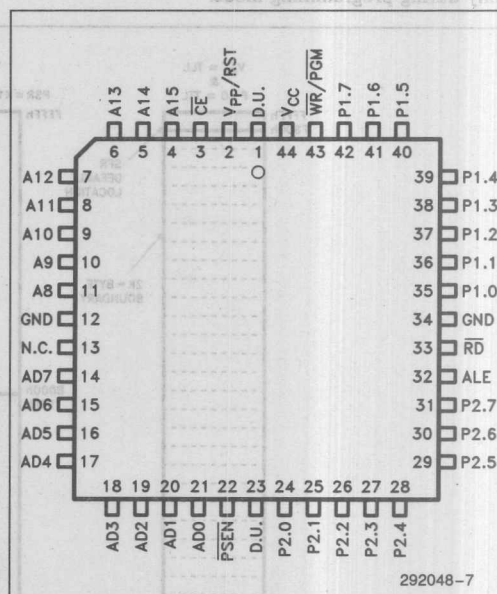


Figure 7. PLCC Pinout.

3 PLANE MEMORY MAP

The 87C75PF has three memory planes: EPROM, SFR, and configuration. Two planes, EPROM and SFR, are available during operating mode. The configuration plane is present under special programming conditions. Figure 8 shows the three memory planes, conditions when they are present, control signals that access them, and memory locations they occupy.

EPROM Plane

The 32K-byte EPROM fills the lower half (0000h-7FFFh default) of the 64K-byte EPROM plane. This conforms to 8051- and 8096-family microcontrollers that have reset and interrupt addresses in the bottom half of the memory map. The EPROM array can adapt to 80188- and 68xx-family microcontrollers by moving it to high memory (8000h-FFFFh). PSEN is the EPROM array's operating- and programming-mode read control. WR/PGM strobes data into the array only during programming mode.

SFR Plane

Special function registers are located in the SFR plane. They occupy low-addresses in a relocatable 2K-byte block (default addresses F800h-FFFFh). The 2K SFR block can be placed on any 2K-byte address boundary to match microcontroller architecture requirements. RD and WR/PGM control reads and writes from/to this plane.

Configuration Plane

The configuration plane contains non-volatile EPROM registers that determine the device's configuration. This plane is available only when high voltages are applied to special pins. PROM programming equipment can use this plane to identify the device, read its present configuration, and program new configurations. Memory-mapped registers can be programmed to:

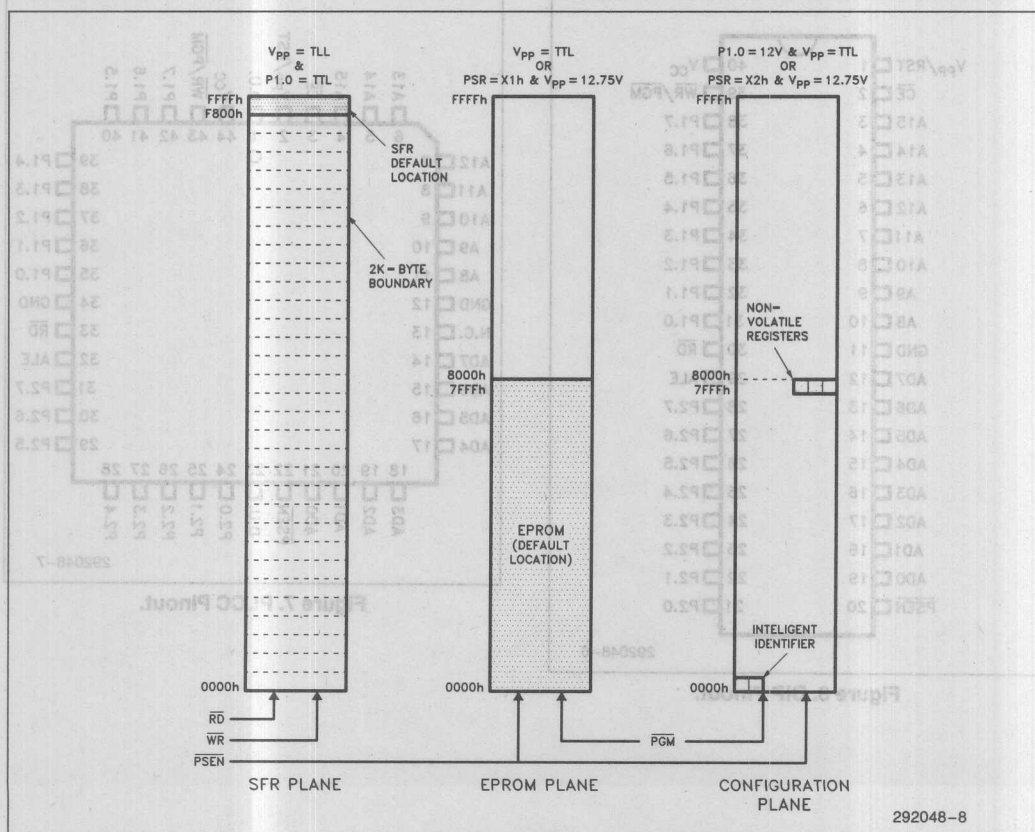


Figure 8. The 87C75PF has three internal memory planes — SFR, EPROM, and Configuration.

- move the EPROM array
- move the SFR block
- combine the EPROM and SFR planes
- combine PSEN and RD
- change RST's polarity
- insert pull-up transistors on port output drivers.

Device reconfiguration will be covered further in the "Architecture compatibility" section.

Plane Select Register

The plane select register (PSR) occupies address F810h in the SFR plane (Figure 9). This register's value determines which plane, EPROM or configuration, is in programming mode. The following plane is programmed when V_{pp} is raised to its programming voltage if PSR contains:

- xxxxxx00 = programming prohibited
- xxxxxx01 = EPROM plane
- xxxxxx10 = configuration plane
- xxxxxx11 = programming prohibited.

Note that both PSR bits must toggle to change planes. Spurious programming noise is unlikely to alter both bits simultaneously. This safeguard prevents erroneous programming of the wrong plane.

I/O PORTS

The 87C75PF has two 8-bit, bi-directional I/O ports. Each port has two addresses in the SFR plane — port latch and port pin. The port latch register drives port pins; it's the port output register. Byte-wide data written to it is strobed by \overline{WR}/PGM 's rising edge. This allows individual register bits to be changed without "glitching" unchanged bits. Port latches can be read to determine previously stored values. Redundant RAM locations that contain port values are not required. Asserting RST sets port latches to "1s".

Each port has a pin register. This input register allows a microcontroller to monitor pin status. Although a port latch register may drive a port pin to "1", an external switch can pull it to "0". A software exclusive-OR of latch and pin values will discover the switch closure.

Figure 9 shows the 2K-byte SFR block (default location shown) containing port addresses. Locations F800h-F807h are reserved for port latch addresses; the 87C75PF uses only two of these addresses. Locations F808h-F80Fh are reserved for port pin addresses; again, the 87C75PF uses only two addresses. Each port latch and port pin register contains eight bits; each corresponding to a port pin. Locations F810h-F81Fh are reserved for SFR registers.

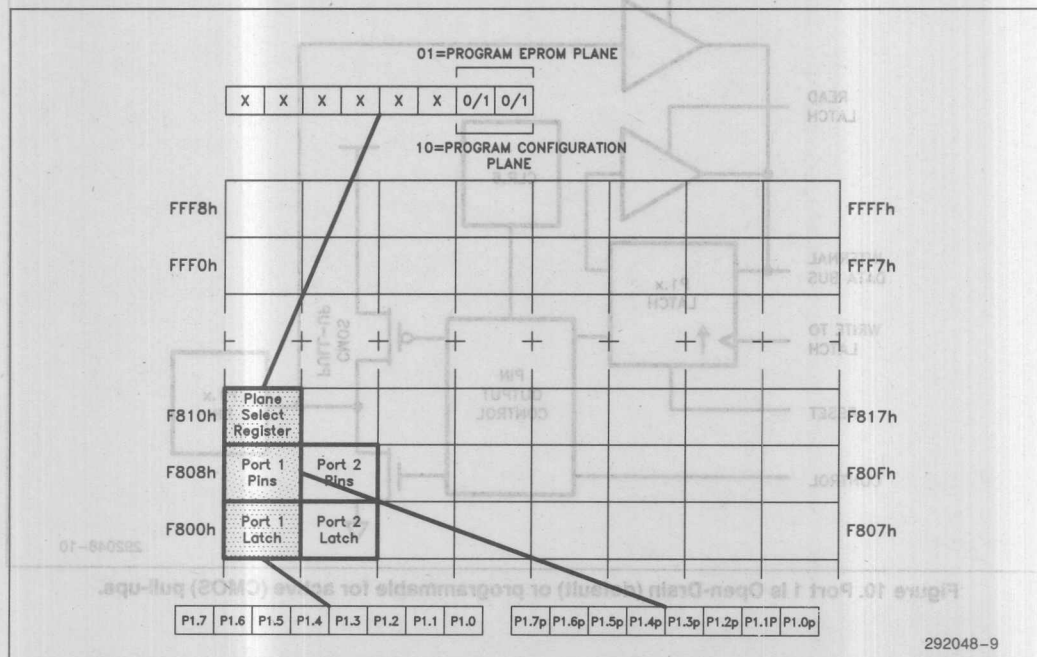


Figure 9. The 2K-byte SFR block contains port latch and pin addresses and Plan Select Register.

Port 1

Port 1's default latch address is F800h; its pin address is F808h. Its default configuration is open drain. Other open-drain devices can be "wire-ORed" to port 1 pins.

Pull-up resistors can be added externally to provide I_{OH} drive.

Port 1's outputs can be reconfigured to supply CMOS drive. Programming the control level register's P1C bit (CLR.6) inserts active pull-up transistors. This switches port 1 pins faster from V_{OL} to V_{OH} and simplifies interfaces to external CMOS devices. Figure 10 shows port 1's block diagram.

Port 2

Port 2 is similar to port 1. Its latch address is F801h and its pin address is F809h. Its default configuration is quasi-bi-directional. This means that each pin has a weak pull-up resistor. External pull-up resistors can be added to increase the port's I_{OH} drive.

Port 2's outputs can be reconfigured to supply CMOS drive. Programming the control level register's P2C bit (CLR.5) inserts active pull-up transistors. Figure 11 shows port 2's block diagram. Note the difference between port 2's and port 1's output stages. In addition to the weak pull-up resistor, the feedback network senses the pin's V_{OH} level and switches a stronger pull-up resistor into the circuit. A V_{OL} level turns the resistor off. Another addition is the pulsed pull-up. When a port latch value changes from "0" to "1", the CMOS transistor is pulsed to quickly supply current to the pin.

ARCHITECTURE COMPATIBILITY

Every microcontroller family has its own architecture. Each has unique boot-up, interrupt, and vectored addresses. Some support dual external memory planes while others communicate with only one. External addressing capacity varies from 64K- to 1M-bytes.

The 8051's control signals and software instructions manipulate 5 memory planes. Three planes are internal — on-chip ROM/EPROM, RAM/SFR, and bit-ad-

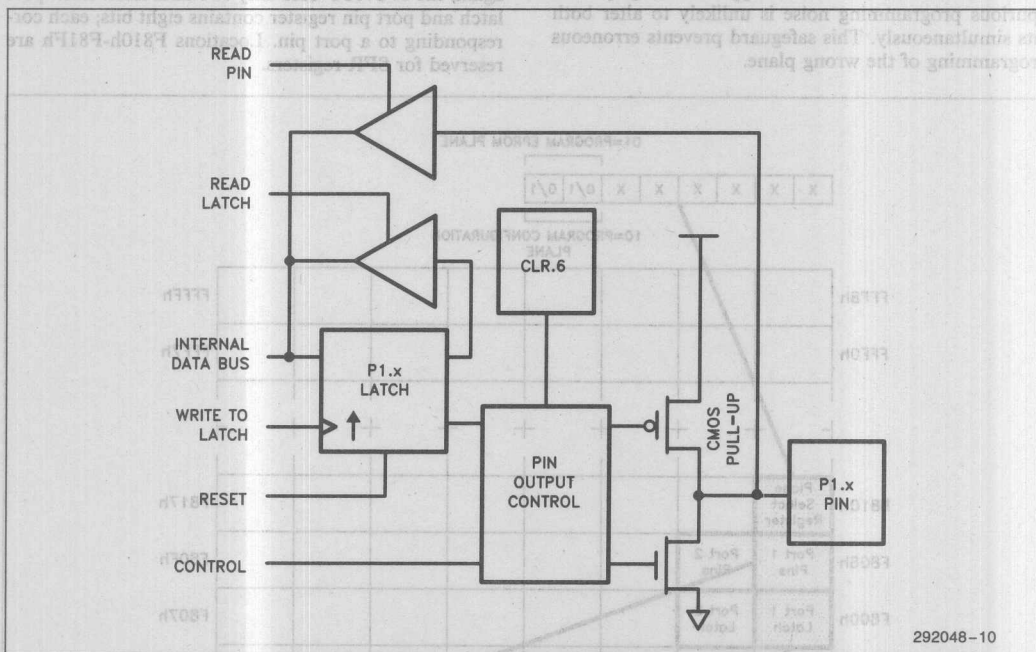


Figure 10. Port 1 is Open-Drain (default) or programmable for active (CMOS) pull-ups.

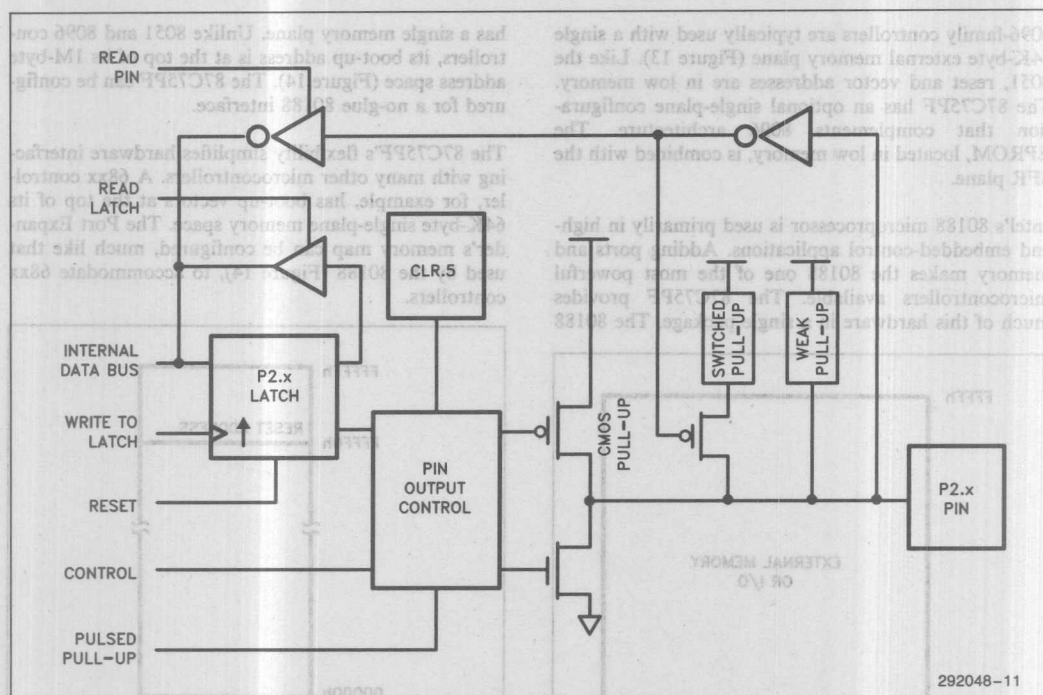


Figure 11. Port 2 is Quasi-bi-directional (default) or programmable for active (CMOS) pull-ups.

dressable registers. Two planes are external — program (EPROM) and data (RAM) memory. The instruction type drives internal and external read, write, and bus signals that select individual planes. An 8051 controller requires non-volatile boot-up memory, internal or ex-

ternal, at the bottom of its program memory plane. The 87C75PF's two-plane, external-memory architecture (see Figure 12) matches the 8051's architecture. EPROM defaults to the EPROM plane's low-memory and SFRs default to the SFR plane's high-memory.

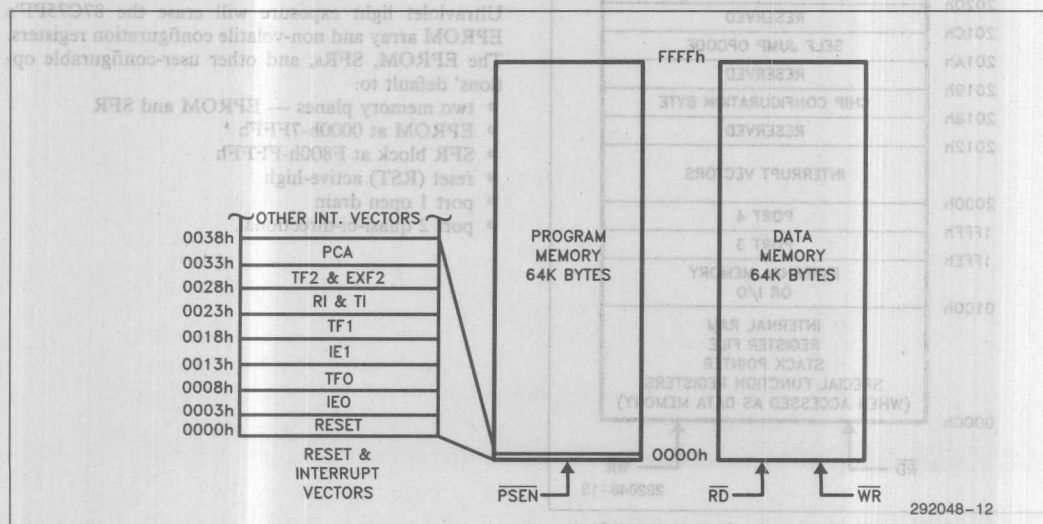


Figure 12. The 8051's two-plane memory has reset and vector addresses in low program-memory.

8096-family controllers are typically used with a single 64K-byte external memory plane (Figure 13). Like the 8051, reset and vector addresses are in low memory. The 87C75PF has an optional single-plane configuration that complements 8096 architecture. The EPROM, located in low memory, is combined with the SFR plane.

Intel's 80188 microprocessor is used primarily in high-end embedded-control applications. Adding ports and memory makes the 80188 one of the most powerful microcontrollers available. The 87C75PF provides much of this hardware in a single package. The 80188

has a single memory plane. Unlike 8051 and 8096 controllers, its boot-up address is at the top of its 1M-byte address space (Figure 14). The 87C75PF can be configured for a no-glue 80188 interface.

The 87C75PF's flexibility simplifies hardware interfacing with many other microcontrollers. A 68xx controller, for example, has boot-up vectors at the top of its 64K-byte single-plane memory space. The Port Expander's memory map can be configured, much like that used by the 80188 (Figure 14), to accommodate 68xx controllers.

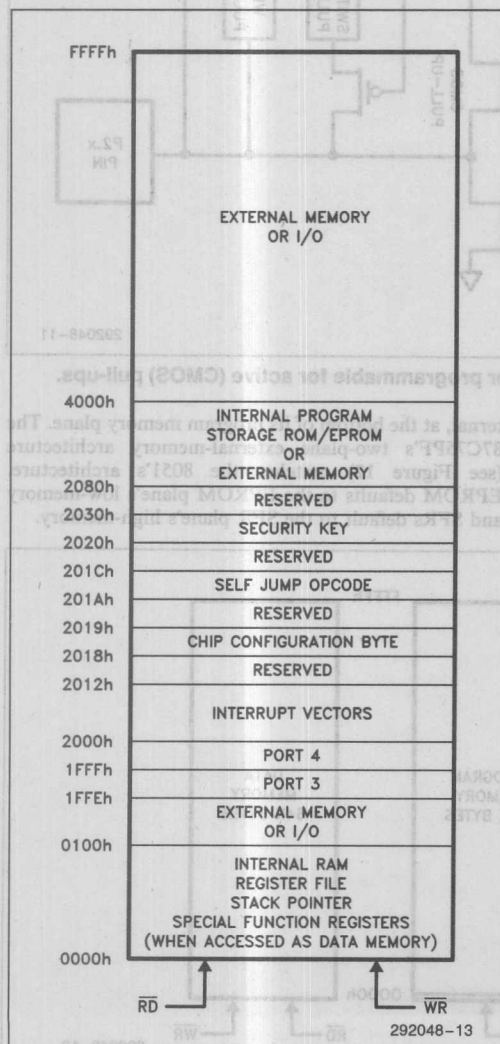


Figure 13. The 8096 has a single memory plane.

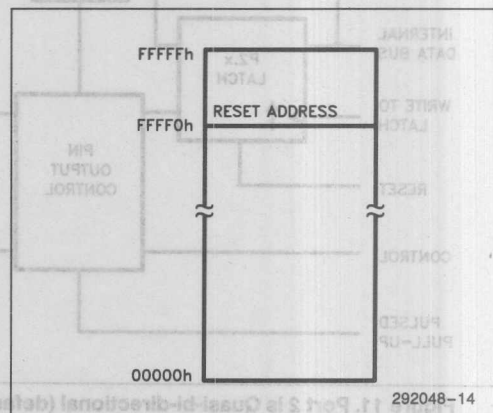


Figure 14. The 80188 boots up at the top of its 1M-byte address space.

Default Configuration

Ultraviolet light exposure will erase the 87C75PF's EPROM array and non-volatile configuration registers. The EPROM, SFRs, and other user-configurable options' default to:

- two memory planes — EPROM and SFR
- EPROM at 0000h-7FFFh
- SFR block at F800h-FFFFh
- reset (RST) active-high
- port 1 open drain
- port 2 quasi-bi-directional.

Changing the Reset Polarity

8051-family microcontrollers have active high reset inputs. 8096, 68xx, 80188, and special 8051-architecture controllers have active-low resets. The 80188 also has an active-high synchronous reset output.

The Port Expander's alterable reset input (RST) can match any microcontroller. When erased, the 87C75PF's RST is active-high. Programming the configuration plane's control level register bit CLR.7 changes RST to active-low (see Figure 15).

Changing Port Output Drive

If port 1 and/or port 2 are used only as outputs, it may be preferable to have CMOS-type output levels. Pro-

gramming CLR.6, P1C, and/or CLR.5, P2C (see Figure 15), inserts active pull-up transistors in port output buffers. These transistors supply higher current and faster switching than open drain or quasi-bi-directional outputs.

Moving the EPROM

The 87C75PF's EPROM can be relocated to the upper half of its 64K-byte memory map. When erased, the EPROM is correctly positioned in low memory for 8051- and 8096-family controllers. Programming the configuration plane's EPROM Location bit, ELR.7 (Figure 16), moves the EPROM to high memory for 80188 and 68xx compatibility.

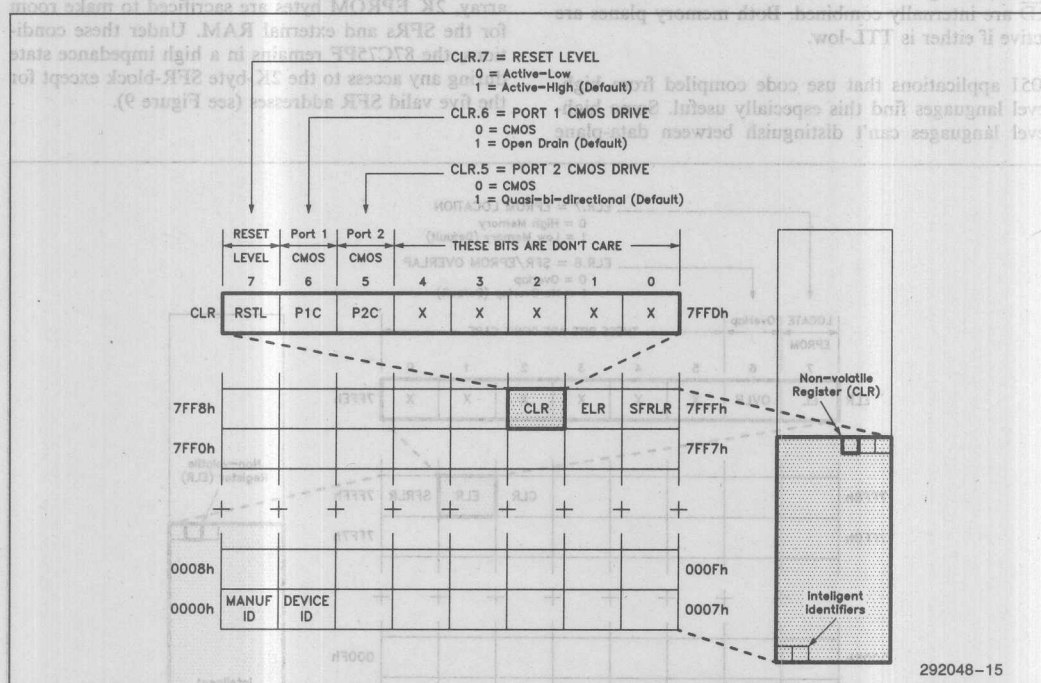


Figure 15. The Control Level Register (CLR) determines the reset pin's polarity and CMOS port drive.

Double- and Single-plane Configurations

The 87C75PF has two operating-mode memory-planes — EPROM and SFR. These planes share identical memory addresses. The EPROM plane is selected when $\overline{\text{PSEN}}$ is TTL-low. The SFR plane is selected when either $\overline{\text{RD}}$ or $\overline{\text{WR}}$ is TTL-low. 8051 microcontrollers use $\overline{\text{PSEN}}$, $\overline{\text{RD}}$, and $\overline{\text{WR}}$ to select two external memory planes. 8096 controllers have only $\overline{\text{RD}}$ and $\overline{\text{WR}}$; some versions have an "INST" output that allows external circuitry to determine when instructions are being issued. Most other microcontrollers provide read and write signals that control only one memory plane.

Programming the 87C75PF's overlap bit, OVLP (ELR.6), converts the device from dual-plane to single-plane (see Figure 16). When ELR.6 = "0", $\overline{\text{PSEN}}$ and $\overline{\text{RD}}$ are internally combined. Both memory planes are active if either is TTL-low.

8051 applications that use code compiled from high-level languages find this especially useful. Some high-level languages can't distinguish between data-plane

and program-plane addresses. For example, look-up tables stored in the same EPROM as program instructions require $\overline{\text{PSEN}}$ to be asserted. However, a compiler interprets look-up table instructions as data fetches. It assigns code that asserts $\overline{\text{RD}}$ instead of $\overline{\text{PSEN}}$. A typical hardware solution uses an AND gate to combine $\overline{\text{PSEN}}$ and $\overline{\text{RD}}$. This forms one memory plane that is accessed by either signal. Programming the 87C75PF's OVLP bit provides this "AND" function.

This bit also permits the SFRs to overlap the EPROM array. This allows multiple Port Expanders to be used in single-plane applications. For example, two Port Expanders can be used in an 8096 system (see Figure 22). Normally, two 87C75PFs' 64K EPROM bytes consume the entire address space leaving no room for port addresses or external RAM. When ELR.6 = "0" and the device's 2K-byte SFR block overlaps its EPROM array, 2K EPROM bytes are sacrificed to make room for the SFRs and external RAM. Under these conditions, the 87C75PF remains in a high impedance state during any access to the 2K-byte SFR-block except for the five valid SFR addresses (see Figure 9).

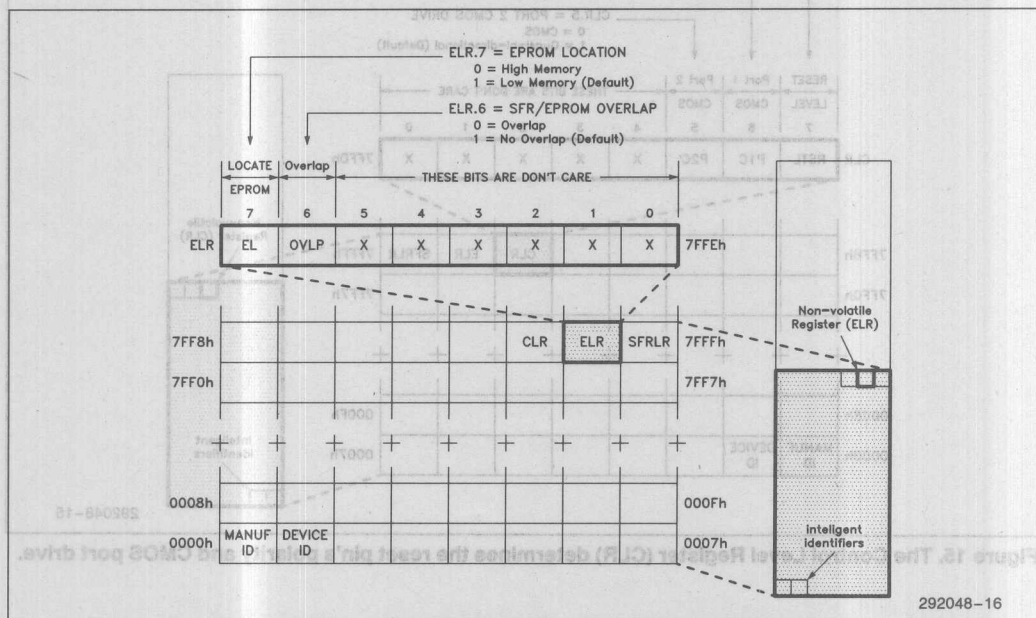


Figure 16. The EPROM Location Register determines the EPROM's memory-map location

The 2K-byte SFR block's default location is F800h-FFFFh in the SFR plane. This location is fine for 8051 and 8096 applications. However, 80188 and 68xx-family controllers have boot-up and vector addresses in this address range; EPROM should be located here.

The SFR block can be moved to any 2K-byte device-address boundary. The SFR location register's (SFRLR) five bits determine the SFR-block's most-significant address bits. When erased, these bits are all "1s", placing the SFRs at 11111xxx xxxxxxxxb or F800h-FFFFh. Programming the SFRLR to 01111xxx, for example, relocates the SFR-block to 7800h-7FFFh (just below the EPROM array when it's at the top of

00000xxx moves the SFRs to the bottom of memory, 0000h-07FFh. Figure 17 shows the SFRLR and its bit definitions.

Programming the Configuration Plane

The 87C75PF data sheet describes detailed programming requirements. PROM programming equipment makes device reconfiguration easy. Down-loading EPROM code (from 0000h to 7FFFh) to the programmer is the same as for any 256K PROM device. The programmer allows editing of CLR, ELR, and SFRLR codes to reconfigure the device. Once programming commences, the EPROM array and the configuration registers are programmed automatically.

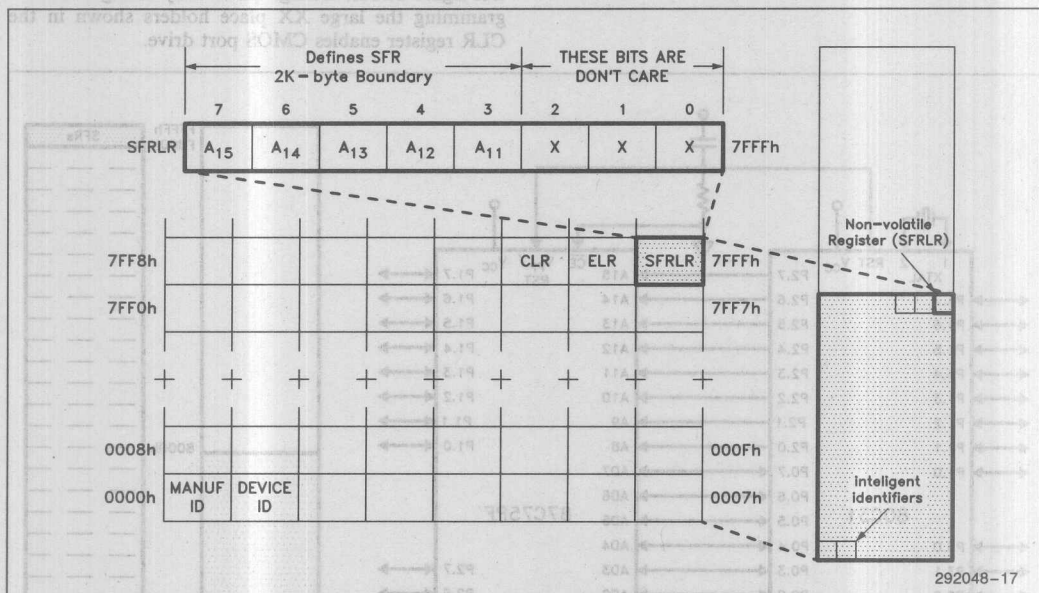


Figure 17. The SFRLR determines the 2K-byte SFR block's base address.

87C75PF APPLICATIONS

Now that you're familiar with how the Port Expander is organized and reconfigured, this section highlights some application examples. You'll see how the 87C75PF connects to 8051, 8096, 80188, and 68xx microcontrollers. Also shown are more sophisticated applications that use multiple Port Expanders and one that allows the microcontroller to program its own Port Expander. All of the applications illustrated show microcontroller/Port Expander interfaces, memory maps, and configuration register (CLR, ELR, SFRLR) values.

80C31 + 87C75PF

8051-family controllers usually operate in two-plane mode. To use external program memory (EPROM) exclusively, the controller's external access pin, EA, is tied to ground. Port 2 supplies upper addresses, A₈–A₁₅. Port 0 becomes the multiplexed lower-address/data bus, AD₀–AD₇. PSEN is the program memory read strobe. WR and RD (port pins P3.6 and P3.7) control external RAM and other read/write devices. RST is active-high on most 8051-family microcontrollers. Some special-purpose '51-based controllers have active-low resets.

Figure 18 shows a typical 80C31 + 87C75PF no-glue application. The 87C75PF's EPROM, SFR, and control-signal default-settings are already configured. Programming the large XX place holders shown in the CLR register enables CMOS port drive.

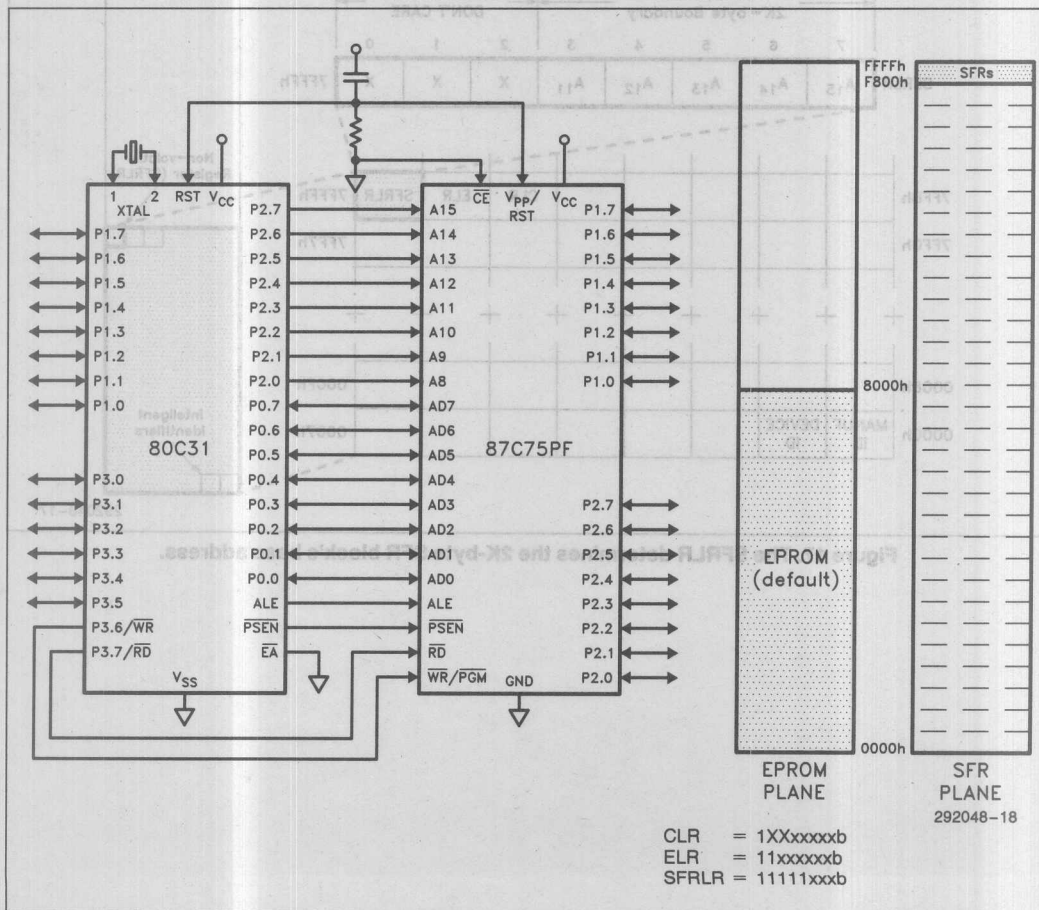


Figure 18. The 87C75PF's no-glue interface takes advantage of the 80C31's two-plane memory map.

80C31 + Two 87C75PFs

High-end applications, such as telecommunications, require sizable program memories and numerous I/O ports. Many of these applications use 8051-family microcontrollers. Two 87C75PF Port Expanders supply added I/O while furnishing EPROM — without using “glue” devices!

Figure 19 shows two Port Expanders in an 80C31 system. Port Expander 1's EPROM is in its default low-memory location (0000h–7FFFh). Its SFR block is moved to F000h, out of Port Expander 2's SFR range (F800h). Port Expander 2's EPROM is moved to high-memory (8000h–FFFFh). Each device's configuration register values are shown below the memory map. This configuration provides 16 additional I/O pins, 64K EPROM bytes, and leaves 60K for RAM and other memory-mapped devices.

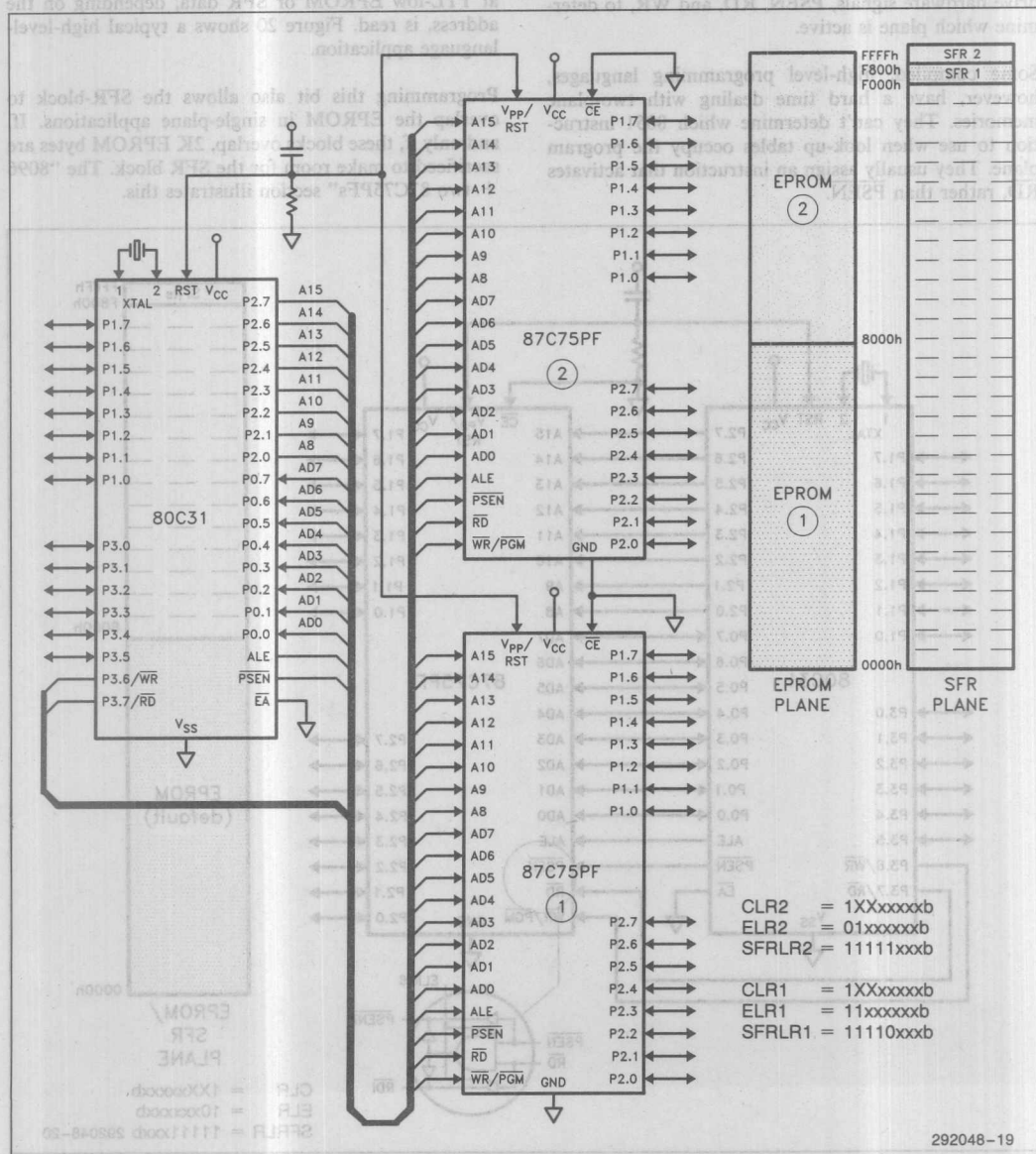


Figure 19. Two 87C75PFs provide 16 I/O pins, 64K EPROM bytes, and room for 60K of RAM.

High-level Language 80C31+ 87C75PF

The 8051's two-plane flexibility challenges hardware and software engineers' creativity. Its two planes logically separate program and data planes to create 128K-bytes of memory in a 64K address space. However, many applications have look-up tables in non-volatile memory, usually in the same EPROM that contains program code. Unique assembly-language instructions drive hardware signals, **PSEN**, **RD**, and **WR**, to determine which plane is active.

Some compiled, high-level programming languages, however, have a hard time dealing with two-plane memories. They can't determine which 8051 instruction to use when look-up tables occupy the program plane. They usually assign an instruction that activates RD, rather than PSEN.

The typical solution forces the system to operate in single-plane mode by combining PSEN and RD with an AND gate. If either signal is TTL-low, the AND gate's output drives a common external-memory read signal. A compiler can now assign its typical "read from data memory" instruction.

The Port Expander has this “AND” function built in. Programming the configuration plane’s Overlap bit, ELR.6, internally combines PSEN and RD; if either is at TTL-low EPROM or SFR data, depending on the address, is read. Figure 20 shows a typical high-level-language application.

Programming this bit also allows the SFR-block to overlap the EPROM in single-plane applications. If, and only if, these blocks overlap, 2K EPROM bytes are sacrificed to make room for the SFR block. The "8096 + two 87C75PFs" section illustrates this.

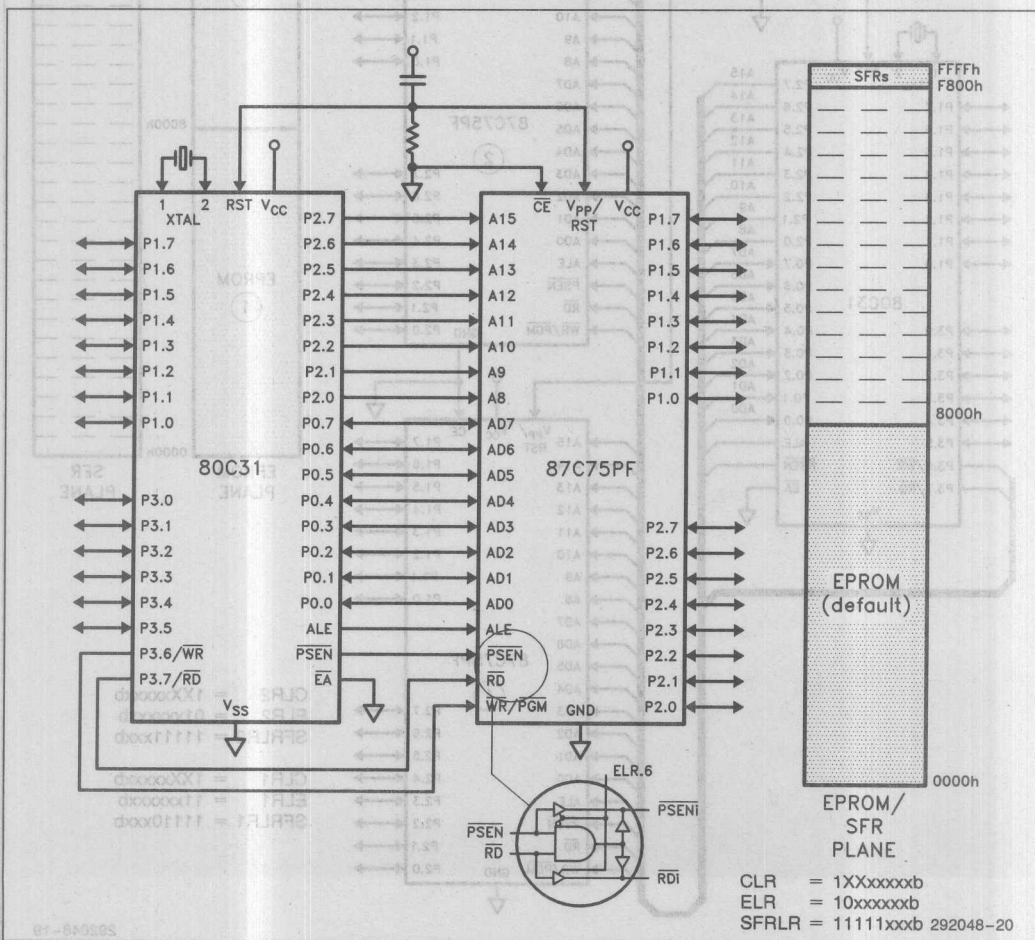


Figure 20. Programming ELR.6 combines $\overline{\text{PSEN}}$ and $\overline{\text{RD}}$ to form a single memory plane.

The 8096 accesses a 64K-byte single-plane memory. Its memory map is similar to the 8051's. External EPROM is required at its low-memory boot-up location (2080h). The 87C75PF's EPROM and SFRs are appropriately located.

The 87C75PF is converted to single-plane mode by either tying PSEN and RD to the 8096's RD pin or by programming ELR.6, the overlap bit. If the latter option is chosen, the unused input, PSEN or RD, should be tied to V_{CC}. Figure 21 shows a "no-glue" 8096 + 87C75PF application.



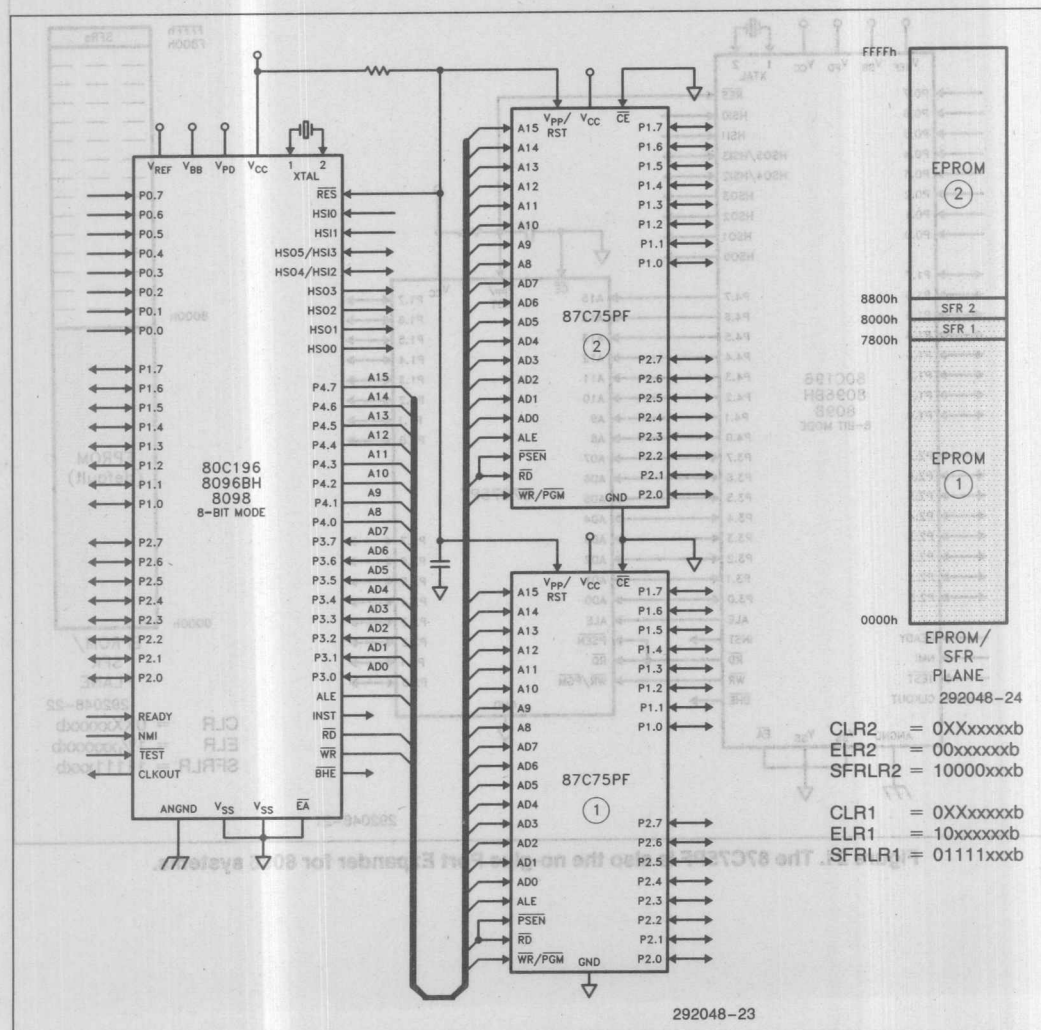
8096 + Two 87C75PFs

Single-plane 8096 applications can use two Port Expanders. Figure 22 shows this no-glue, three-chip system.

Port Expander 1 has its EPROM in default low-memory. Its SFR block is mapped over its EPROM; location 7800h is arbitrarily chosen. Programming 0111xxxxb into SFRLR moves the SFR block. Programming ELR.6 (to "0") overlaps the EPROM and SFR planes; one plane is formed. This bit also tells the Port Expander that its SFRs are intentionally mapped over its EPROM, like Port Expander 1. This configuration supplies four additional 8-bit ports, 60K EPROM bytes, and still leaves 4K bytes free for RAM.

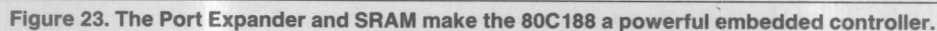
byte block, except valid port and PSR addresses, places the external data bus in a high impedance state. External RAM can occupy the 2K-byte space.

Port Expander 2 is also reconfigured. Its EPROM is moved to high-memory by programming ELR.7. Its SFR block must overlap its EPROM array; 8000h is arbitrarily chosen. Port Expander 2's overlap bit, ELR.6, is programmed to form a single plane and to tell the device that its SFRs are intentionally mapped over its EPROM, like Port Expander 1. This configuration supplies four additional 8-bit ports, 60K EPROM bytes, and still leaves 4K bytes free for RAM.



EPROM-block addresses, (F7800h is shown). Programming the overlap bit, ELR.6, or tying PSEN and RD to the 80C188's RD combines the EPROM and SFR planes. The processor's UCS, connected to the 87C75PF's CE, selects the Port Expander in the upper address range. The 80C188's reset input, RES, is active low. Programming the 87C75PF's RSTL bit, CLR.7, converts RST to active-low. The 80C188 also has an active-high synchronous reset output. This output can be connected to the 87C75PF's RST without reconfiguring RST's polarity.

80C188 systems usually have larger RAM arrays than typical microcontroller applications. Figure 23 shows the simple RAM interface. The RAM does not contain its own address latches, so an 8-bit latch must be used to capture addresses A_0 – A_7 .



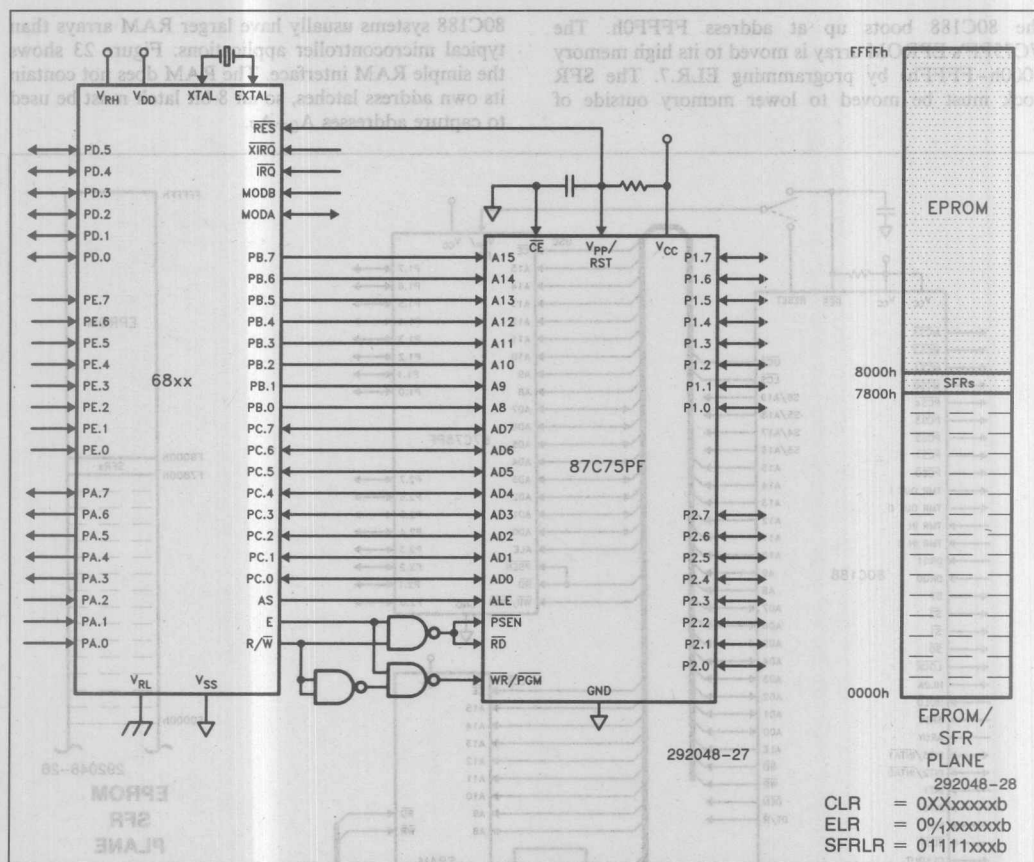
68xx + 87C75PF

The microcomputer industry's peripheral- and memory-interface standard dictates chip-enable, output-enabled, and write-enable polarities. All are active-low. The 87C75PF conforms to this industry standard.

Like Intel controllers, 68xx-family microcontrollers use multiplexed address/data pins. However, they differ in two significant ways. First, 68xx controllers have high-memory reset- and interrupt-vector addresses. Address

A₁₅ is logic-high during vector accesses. Second, read and write controls are functions of R/W and E (clock output). Combinational logic must convert R/W and E to industry-standard RD and WR signals.

The 87C75PF's memory map can be reconfigured and its two memory planes combined to simplify 68xx interfaces. Its RST polarity can match a 68xx's active-low reset. All that's required to complete the interface is to condition R/W and E to RD and WR. Figure 24 shows a 68xx + 87C75PF system and its memory map.



Multiple-application modules can be customized using in-system programming. For example, a generic control module can be built, installed in a variety of end products, and customized for different tasks at the end of the production sequence.

Figure 25 shows a simple 80C51-based in-system-programmable module. The microcontroller's on-chip ROM or EPROM contains the communication and programming algorithms. Port pins P3.0 and P3.1 provide the serial communication link. P3.2 (EACONT) controls the EA pin. When high (which occurs at reset or when "1" is written to it), internal program memory supplies code. When low, external EPROM supplies code. P3.4 (ALECONT) controls the ALE latching signal during programming. P3.5 (PGMON) controls programming and operating-mode V_{pp} and V_{CC} voltages. P3.6, which is the \overline{WR} signal during normal operation, serves as the program pulse strobe, PGM, during programming. RD, P3.7, or PSEN can be used to verify programmed data whenever V_{pp} is at its programming voltage.

Figure 26 shows the program and latch control circuit. 5 volt and 12 volt supplies are connected to this circuit at all times. Inverter 74'06a allows 12 volts to pass into the DC/DC converter and the LM317 voltage regulators only when system power is on. \overline{PGMON} is high after reset or when P3.5 contains a "1." \overline{PGMON} controls inverter 74'06b which turns V_{pp} on or off. Inverter 74'06c keeps V_{CC} at 5 volts until programming commences. When \overline{PGMON} goes low, these inverters turn off allowing V_{pp} and V_{CC} voltages to attain their programming levels. The variable resistors adjust V_{pp} and V_{CC} read- and program-voltages. V_{CC} read voltage is 5.0V and its program voltage is 6.25V. V_{pp} read voltage is off, so it doesn't interfere with the 87C75PF's reset, and its program voltage is 12.75V.

\overline{PGMON} also controls the ALE circuit. When \overline{PGMON} is high, the microcontroller's ALE value passes to the 87C75PF's ALE pin. When \overline{PGMON} is low, the microcontroller's ALECONT controls ALE.

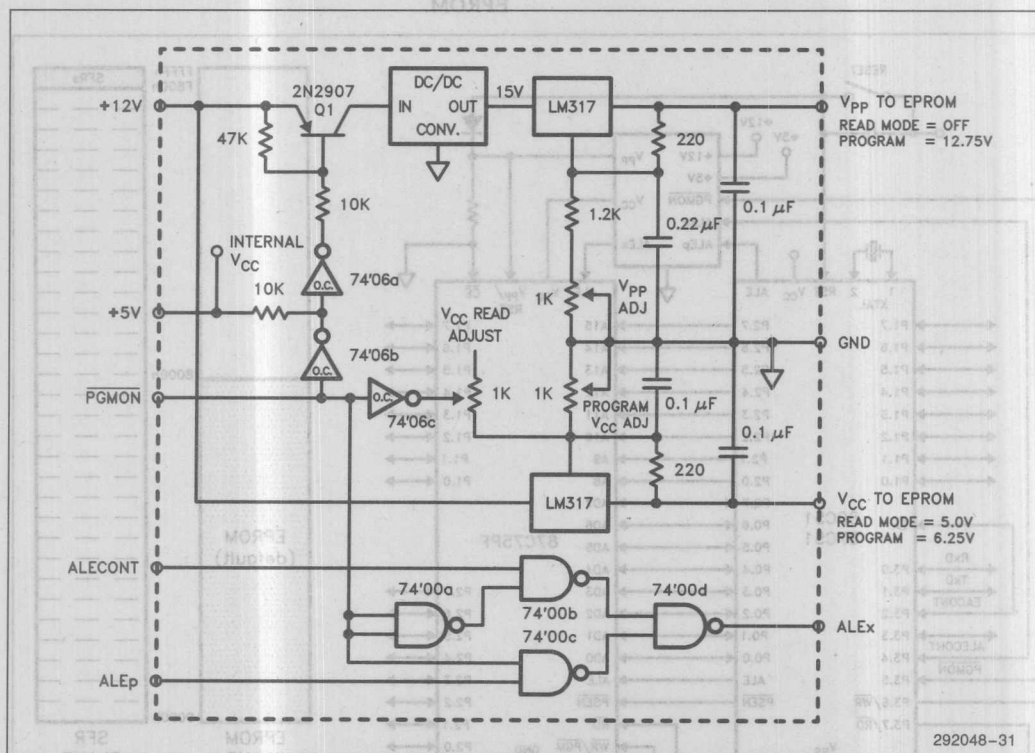


Figure 26. A microcontroller can use this circuit to control programming voltages and ALE.

The microcontroller's AD₀₋₇ and A₈₋₁₅ (ports 0 and 2) connect to the 87C75PF's AD₀₋₇ and A₈₋₁₅ pins. The controller's program-memory read signal, PSEN, controls the 87C75PF's output-enable, PSEN.

During programming, the controller brings EACONT high and PGMON low. This allows it to operate from internal code, enables programming voltages on the 87C75PF's V_{pp} and V_{CC} pins, and switches ALE control from the controller's ALE to its ALECONT. It then inputs data over its serial channel. With ALECONT high, an address is placed on ports 0 and 2. When ALECONT is brought low, the 87C75PF internally latches the address. Data read from the serial port is written to port 0. Port 0 must have pull-up resistors when used in its I/O port mode. The Port Expander now has both address and data information. The controller needs only to bring its WR pin low to program data into the addressed location.

The in-system programming sequence is summarized below.

- 1) Set EACONT = "1". Code is now supplied from the controller's internal program memory.
- 2) Assert PGMON. This switches V_{pp} and V_{CC} to their program voltages and allows the controller to manually control ALE via ALECONT. ALECONT and WR are high.
- 3) Download address and data information via Port 3's serial channel. Ports 0 and 2 serve as I/O ports, so place the 16-bit address on them. Bring ALECONT low to latch the address into the 87C75PF.
- 4) Write data information to port 0.
- 5) Bring WR low to program data into the 87C75PF. See the 87C75PF data sheet for the programming algorithm and timing requirements.

- 6) Verify the programmed data. When the 87C75PF's V_{pp} is at 12.75V, its PSEN and RD pins are internally combined. The "MOVC A,@A+DPTR" instruction uses the PSEN pin to read EPROM data (or the "MOVX A,@DPTR" instruction uses the RD pin).
- 7) Repeat this sequence until all EPROM data is programmed and verified.
- 8) When programming is complete, de-assert PGMON and ALECONT. When EACONT = "0", code execution commences from the 87C75PF. Code duplication at identical internal and external memory locations allows uninterrupted paging between these two memory spaces.

When 6.25V is applied to the 87C75PF's V_{CC} during programming, its port outputs, when "1", will be close to 6.25V. Careful system design should ensure that microcontroller and other device inputs can handle this elevated voltage. Writing "0s" to all port pins before V_{CC} receives 6.25V will prevent damage to external devices.

SUMMARY

System demands push single-chip microcontroller designs to their limits. Complex applications are I/O intensive and use lots of EPROM. Traditional solutions use discrete chips – EPROM, address latches, address decoders, I/O port chips, and "glue" logic – to get more memory and expand, or recover, I/O.

Intel's 87C75PF Port Expander puts port functions, EPROM, and "glue" into a single package. Chip count and board size are dramatically reduced. System performance is optimized. Reliability is assured. Design time is shortened. Manufacturing is simplified. Device inventory is reduced.

Miniaturized system designs that weren't possible before, can now come to life, thanks to the 87C75PF.

TERRY KENDALL

MICROCONTROLLER PERIPHERALS

Verify the programmed data. When the 87C325P's Vpp is at 12.75V, its PSEN and RD pins are internally combined. The "MOVX A,@A+DPTR" instruction uses the PSEN pin to read EPROM data (or the "MOVX A,@DPTR" instruction uses the RD pin).

7) Repeat this sequence until all EPROM data is programmed and verified.

8) When programming is complete, de-assert P0MON and ALECONT. When EACONT="0", code execution commences from the 87C325P. Code duplication at identical internal and external memory locations allows uninterrupted paging between these two memory spaces.

When 6.25V is applied to the 87C325P's Vcc during programming, its port outputs, when "1", will be close to 6.25V. Careful system design should ensure that the microcontroller and other device inputs can handle this elevated voltage. Writing "0s" to all port pins before Vcc receives 6.25V will prevent damage to external devices.

SUMMARY

System demands push single-chip microcontroller designs to their limits. Complex applications are I/O intensive and use lots of EPROM. Traditional solutions require external logic, latches, address buffers, and more memory and expansion. I/O, however, is reduced.

Minimized system designs that weren't possible before can now come to life, thanks to the 87C325P. EPROM, and "glue" into a single package. Chip count and board size are dramatically reduced. System performance is optimized. Reliability is assured. Design time is shortened. Manufacturing is simplified. Device inventory is reduced.

Minimized system designs that weren't possible before can now come to life, thanks to the 87C325P.

The microcontroller's AD0-7 and A8-15 (ports 0 and 2) connect to the 87C325P's AD0-7 and A8-15 pins. The controller's programming read signal, PSEN, controls the 87C325P's output-enable, PSEN.

During programming, the controller brings EACONT high and P0MON low. This allows it to operate from internal code, enables programming voltages on the 87C325P's Vpp and Vcc pins, and switches ALE control from the controller's ALE to its ALECONT. It then inputs data over its serial channel. With ALECONT high, an address is placed on ports 0 and 2. When ALECONT is brought low, the 87C325P internally latches the address. Data read from the serial port is written to port 0. Port 0 must have pull-up resistors in its I/O port mode. The Port Expander outputs read in its address and data information. The controller needs only to bring its WR pin low to program data into the addressed location.

The in-system programming sequence is summarized below.

- 1) Set EACONT="1". Code is now supplied from the controller's internal program memory.
- 2) Assert P0MON. This switches Vpp and Vcc to their program voltages and allows the controller to manually control ALE via ALECONT. ALECONT and WR are high.
- 3) Down-load address and data information via Port 2's serial channel. Place the address on the ALECONT pin to latch the address into the 87C325P.
- 4) Write data into the addressed location.
- 5) Bring WR low to program data into the 87C325P. See the 87C325P data sheet for the programming algorithm and timing requirements.

July 1988

Latched EPROMs Simplify Microcontroller Designs

TERRY KENDALL
MICROCONTROLLER PERIPHERALS

INTRODUCTION

Board Space. Simplified design. Reliability. Manufacturability. Performance. Cost. Designers balance these requirements in every project, especially in microcontroller applications.

This application note will show how Intel's latched EPROMs minimize board space and cost, simplify design and manufacturing, and increase performance and reliability in microcontroller systems.

A few years ago an embedded control system consisted of many discrete components. A general purpose microprocessor was combined with memories, timers, counters, I/O expanders, address decoders, latches, and assorted glue chips to make a basic control system. Then came the microcontroller. These functions, and many more, are now combined into a single chip.

Today, engineers are stretching the limits of microcontroller features. Controller manufacturers are stuffing as many functions and as much memory as die and package can accommodate. Microcontrollers typically have EPROM (or ROM) densities of 4K or 8K bytes; some advanced controllers even have 16K. Still, more is required.

Microcontroller applications are now moving back to multiple chip solutions. 32K-byte EPROMs are common in many medium and high-end systems. It is not

practical to put this much memory on the microcontroller die; chip price becomes prohibitive. Most controllers have an expansion mode that allows external memory to be added.

Higher density is not the only reason to go "off-chip" for memory. Many systems are designed to be generic modules. For example, one engine control module can be designed for an entire line of car models. During a final manufacturing step the module can be custom programmed for any particular vehicle. ROM-version controllers don't lend themselves to this application. EPROM memory allows any application to be customized — at any step in the manufacturing process.

But, using off-chip memory shouldn't detract from the designer's goal to achieve a minimum-chip system. Latched EPROMs provide microcontroller memory expansion without adding "glue" chips.

THE MULTIPLEXED BUS

To achieve small board space, embedded control systems require not only minimum chip count but chips that occupy small footprints. Embedded controllers achieve this by using multiplexed address/data buses. An 8051 controller, for example, shares its lower eight address pins with its 8-bit data.

Every memory access requires two cycles — one for address, one for data (see Figure 1). The controller

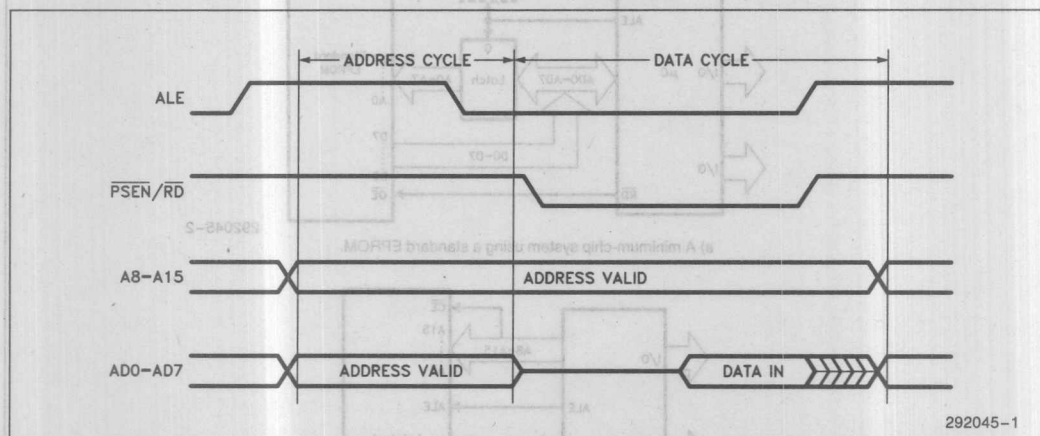


Figure 1. Every microcontroller memory access requires two cycles.

places a 16-bit address on the bus during the first cycle. It holds the upper eight bits constant throughout the access. It presents the lower address byte just long enough for an external latch to capture it. The latch and controller's upper bus supply the address to external devices for the remainder of the memory access. The controller uses its multiplexed lower address/data pins to transmit or receive data during the data cycle. As well as minimizing the controller's pin count, the multiplexed bus requires fewer board traces.

Before latched EPROMs, adding external memory to microcontrollers consumed excess board space. Address latches plus EPROM required more space than the controller itself. The address latch consumes significant board space and system power, degrades system reliability and EPROM performance, and complicates design and manufacturing.

Intel's high-performance latched EPROMs don't compromise designers' goals to produce minimum chip systems. The address latching function is built into the EPROM chip. The no-glue controller-EPROM interface simplifies design and manufacturing while increasing performance and reliability — in the smallest possible board space.

MICROCONTROLLER MEMORY INTERFACE

A typical microcontroller/memory interface is shown in Figure 2. Eight-bit controllers require at least one 8-bit address latch; Sixteen-bit controllers require two. In an 8-bit system, the controller's A₈₋₁₅ address pins are connected directly to the EPROM's upper address pins. Address/data pins AD₀₋₇ are connected to the EPROM's D₀₋₇ data pins and to the address latch's inputs. The latch's outputs drive the EPROM's A₀₋₇ address inputs. The controller's address-latch-enable, ALE, controls the latch. Figure 2a shows this memory interface.

Figure 2b shows a simplified system that uses a latched EPROM. All of the controller's bus signals connect directly to the latched EPROM. It's easy to see that design time (and manufacturing) are simplified. Performance is improved because latch propagation delay is non-existent. System reliability is assured — one factory-tested, integrated memory device is inherently more reliable than several discrete components.

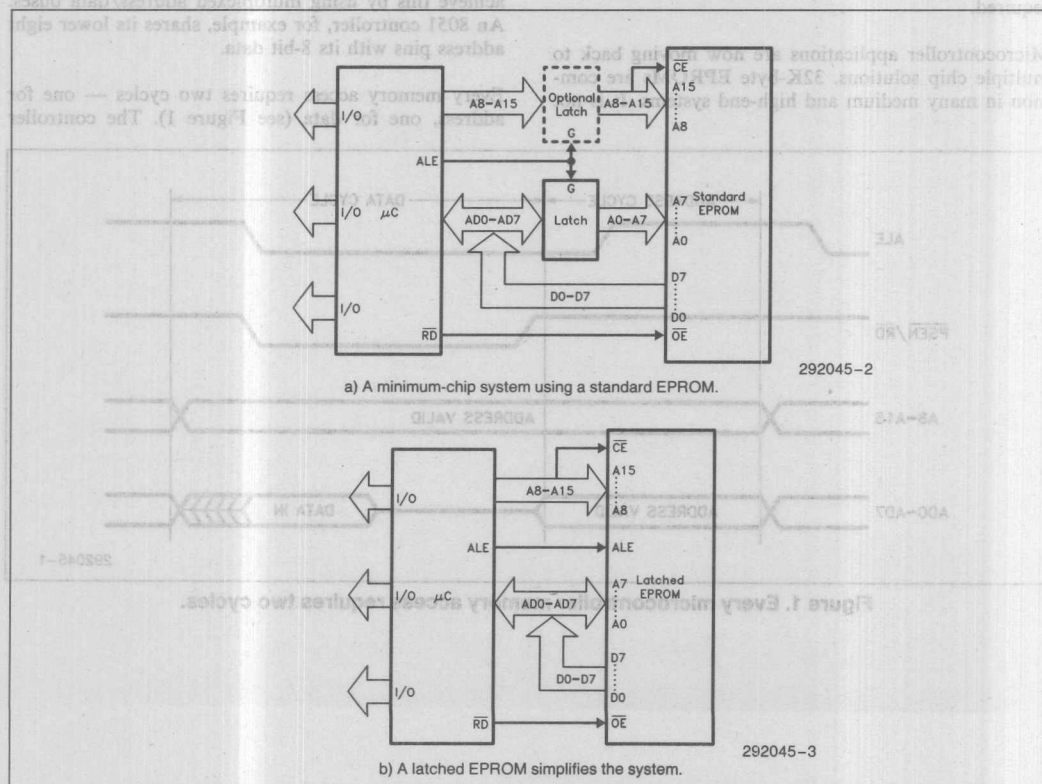


Figure 2. Typical microcontroller/memory systems are improved with latched EPROMs.

Discrete latch chips, like the 74HCT573, have large output drivers. This allows them to drive many devices on a system's address bus. Unfortunately, large output drivers consume considerable power. Typical microcontroller applications are minimum-chip systems. Discrete address latches unnecessarily waste system power with their large drive capability. Intel's latched EPROMs use very little power because their built-in latches drive only internal address lines. Integrated address latches allow "no-glue" interfacing to 8-bit and 16-bit microcontrollers.

SYSTEM INTEGRITY

An address latch and associated board traces require about .75 inches². This doesn't sound like much, but compared to the EPROM's 1.2 in² and the controller's 1.5 in² it amounts to 22% of a system's board space.

Not only does a latched EPROM produce a more "elegant" design, but system reliability is improved. Every

board component is subject to failure. A discrete latch requires twenty additional PC-board solder joints — each a potential failure point. Failures decrease as part count (elimination of latches) goes down.

Every board trace and component node is a source (or receptor) of system noise. Noise can degrade performance and compromise data integrity. EPROM performance requires rock-steady address inputs. When EPROM output buffers turn on, address input buffers are affected. A small ground reference fluctuation changes the threshold voltage of input buffer transistors. This can effectively change the EPROM's address in mid-access; data integrity is compromised.

Latched EPROMs are virtually immune to ground-reference shifts. Current surge caused by switching output buffers may affect the EPROM's address inputs, but the internally latched address remains steady; noise isn't transferred to address decoders. Access time and data integrity are optimized.

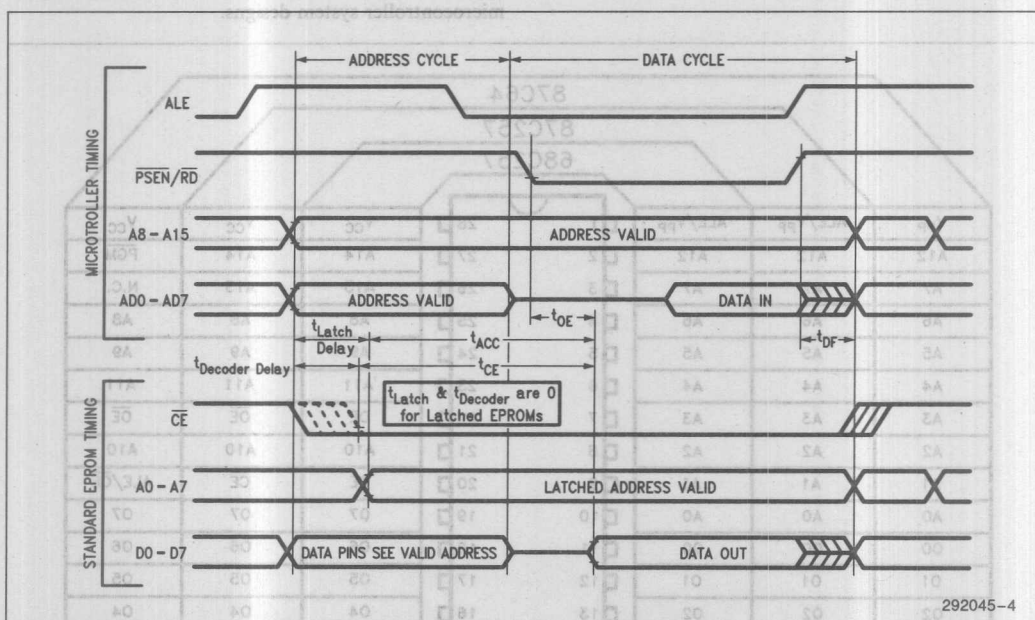


Figure 3. Propagation delays can be significant when standard EPROMs are used in uC systems. Latched EPROMs eliminate these delays.

SYSTEM PERFORMANCE

Latched EPROMs improve system performance. Discrete latches have inherent propagation delays. In a pure CMOS system, this delay is significant; a 74HCT373 latch delay is 45ns at automotive and military temperatures. A 16MHz 80C31 microcontroller, for example, provides 207ns for EPROM access time. A 45ns latch delay degrades this access time to 162ns. An EPROM rated at 160ns or faster must be used. Figure 3 shows the timing delays inherent in discrete component solutions.

If a latched EPROM is used, no external latch delay occurs. A 200ns latched EPROM can be used. Access time parameters include internal latch propagation delays. Slower, less expensive latched EPROMs do the same job as fast EPROMs and discrete latches.

ARCHITECTURE COMPATIBILITY

Intel's latched EPROMs have separate address and data pins. All address inputs contain latches. This simplifies 16-bit microcontroller interfacing. Pin layout is virtually identical to standard EPROMs. Upgrade-compatible circuit board designs are simplified. In 8-bit multiplexed address/data systems, EPROM pins A₀₋₇ are connected directly to corresponding D₀₋₇ pins. In 16-bit multiplexed systems, low-byte EPROM data pins D₀₋₇ are connected to address lines A₀₋₇ while high-byte EPROM data pins D₀₋₇ are connected to address lines A₈₋₁₅. See Figures 7 and 9 for typical 8-bit and 16-bit system examples.

THE LATCHED EPROM FAMILY

Intel's growing family of latched EPROMs includes the 87C64, 87C257, and 68C257. Ceramic DIP and PLCC package pinouts are shown in Figures 4 and 5. This application note shows how latched EPROMs simplify microcontroller system designs.

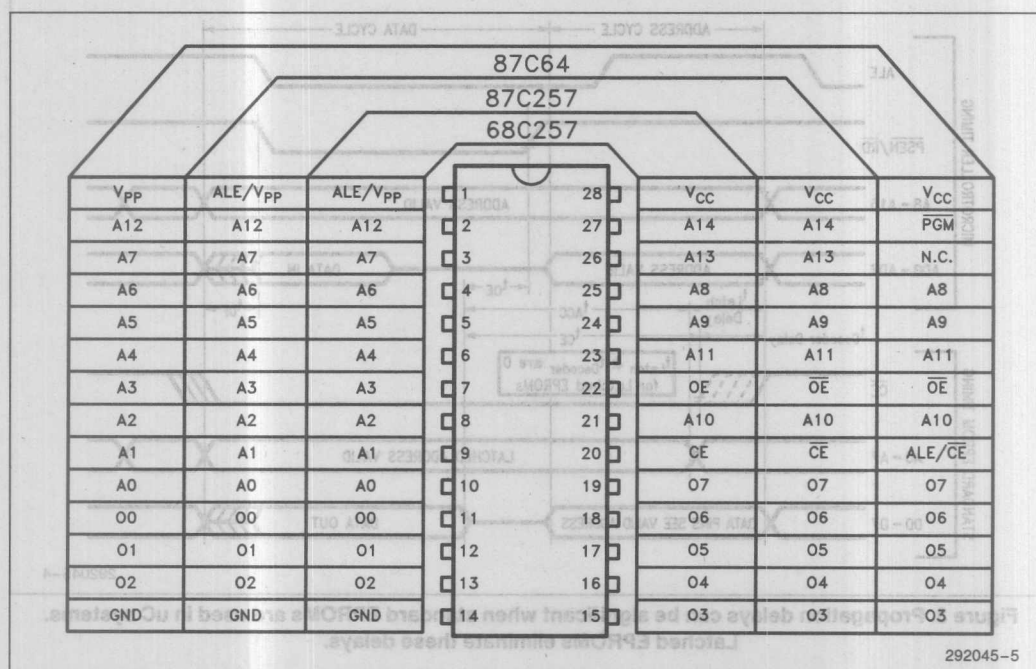


Figure 4. 28-pin ceramic DIP latched-EPROM pinouts

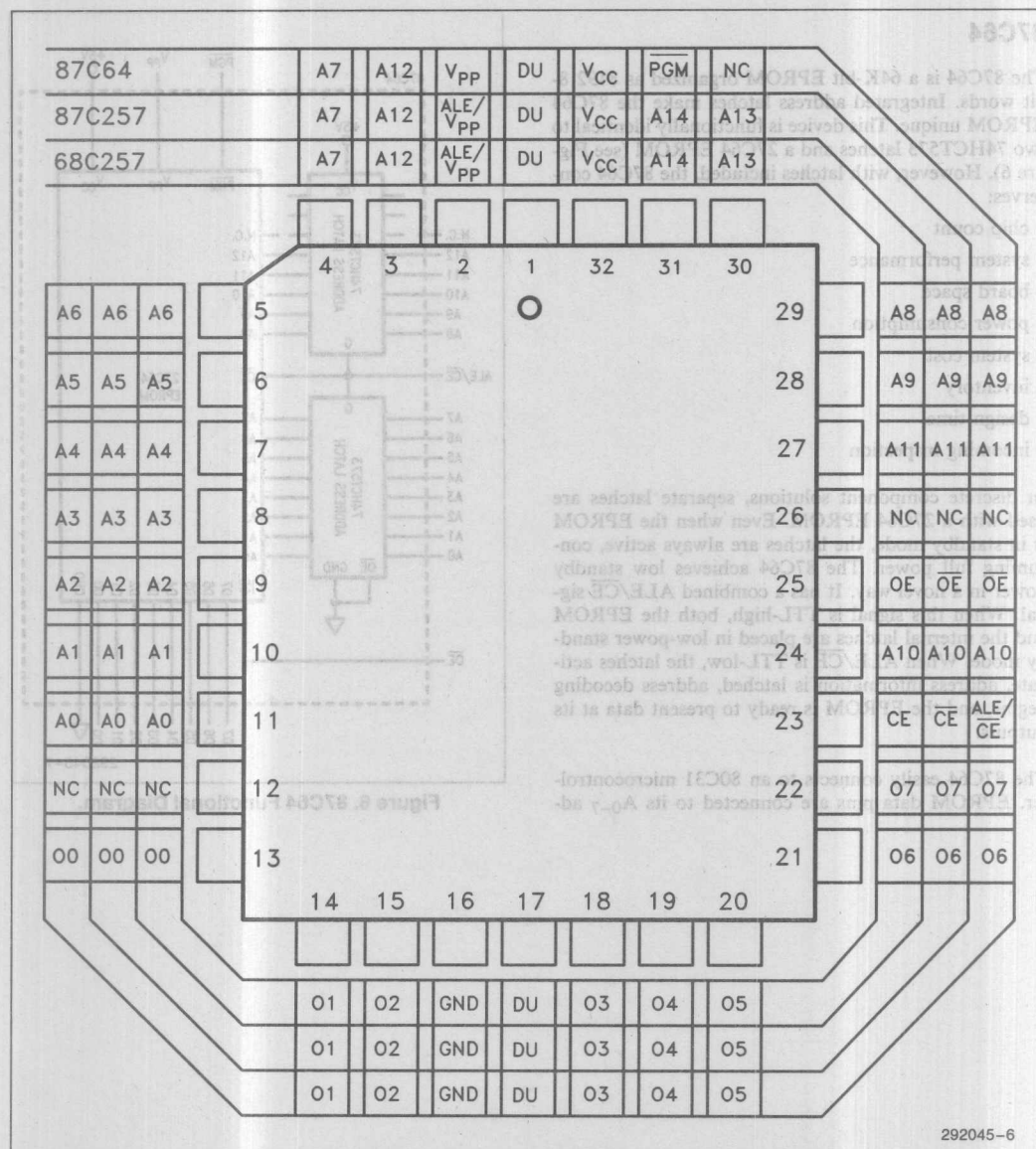


Figure 5. 32-Lead PLCC latched-EPROM pinouts.

87C64

The 87C64 is a 64K-bit EPROM organized as 8192 8-bit words. Integrated address latches make the 87C64 EPROM unique. This device is functionally identical to two 74HCT573 latches and a 27C64 EPROM (see Figure 6). However, with latches included, the 87C64 conserves:

- chip count
- system performance
- board space
- power consumption
- system cost
- inventory
- design time
- incoming inspection

In discrete component solutions, separate latches are used with a 27C64 EPROM. Even when the EPROM is in standby mode, the latches are always active, consuming full power. The 87C64 achieves low standby power in a novel way. It has a combined ALE/CE signal. When this signal is TTL-high, both the EPROM and the internal latches are placed in low-power standby mode. When ALE/CE is TTL-low, the latches activate, address information is latched, address decoding begins, and the EPROM is ready to present data at its outputs.

The 87C64 easily connects to an 80C31 microcontroller. EPROM data pins are connected to its A₀₋₇ ad-

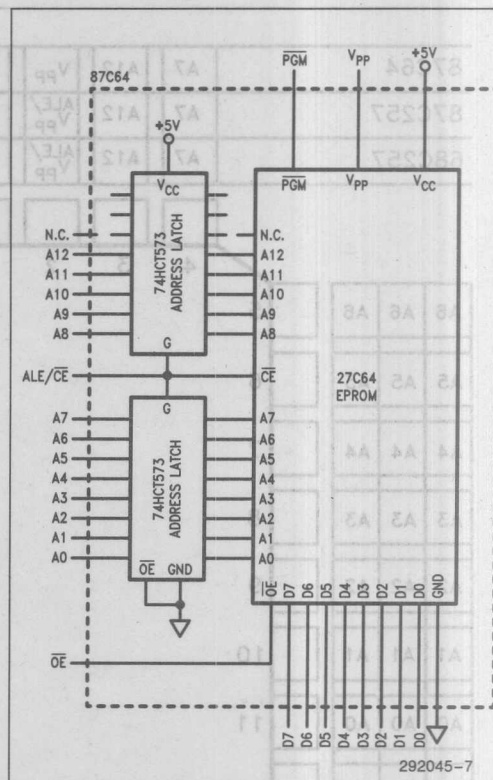


Figure 6. 87C64 Functional Diagram.

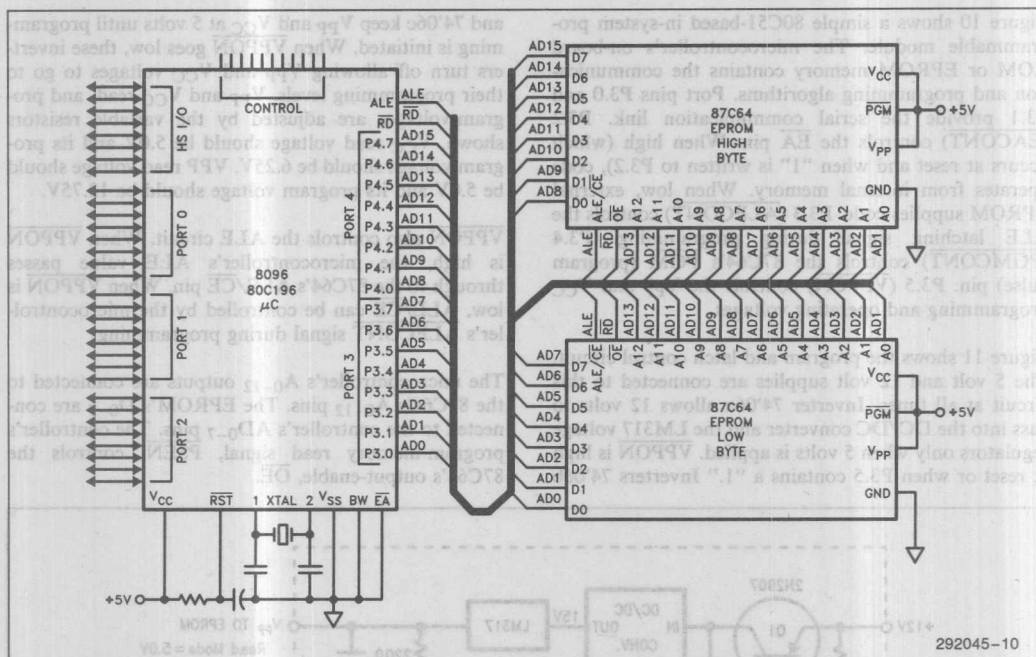


Figure 9. Two 87C64s provide a no-glue EPROM solution for word-wide systems.

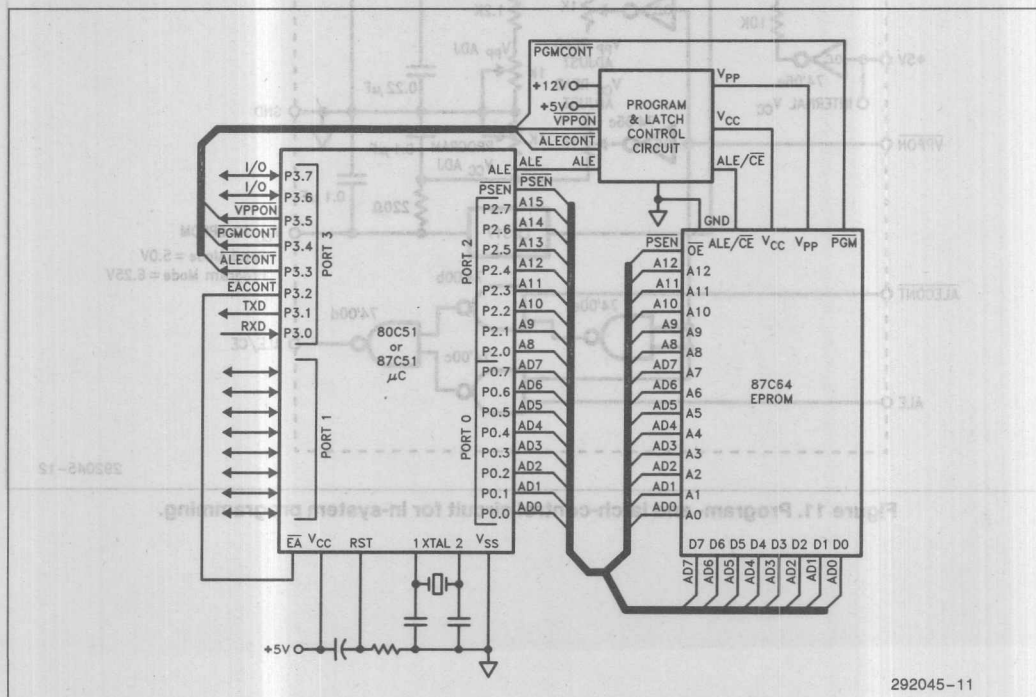


Figure 10. A simple in-system programmable module.

Figure 10 shows a simple 80C51-based in-system programmable module. The microcontroller's on-board ROM or EPROM memory contains the communication and programming algorithms. Port pins P3.0 and P3.1 provide the serial communication link. P3.2 (EACONT) controls the \overline{EA} pin. When high (which occurs at reset and when "1" is written to P3.2), code operates from internal memory. When low, external EPROM supplies code. P3.3 (ALECONT) controls the ALE latching signal during programming. P3.4 (PGMCONT) controls the 87C64's PGM (program pulse) pin. P3.5 (VPPON) controls the V_{pp} and V_{CC} programming and operating voltages.

Figure 11 shows the program and latch control circuit. The 5 volt and 12 volt supplies are connected to this circuit at all times. Inverter 74'06a allows 12 volts to pass into the DC/DC converter and the LM317 voltage regulators only when 5 volts is applied. V_{PPON} is high at reset or when P3.5 contains a "1." Inverters 74'06b

and 74'06c keep V_{pp} and V_{CC} at 5 volts until programming is initiated. When V_{PPON} goes low, these inverters turn off allowing V_{pp} and V_{CC} voltages to go to their programming levels. V_{pp} and V_{CC} read- and program-voltages are adjusted by the variable resistors shown. V_{CC} read voltage should be 5.0V and its program voltage should be 6.25V. V_{pp} read voltage should be 5.0V and its program voltage should be 12.75V.

V_{PPON} also controls the ALE circuit. When V_{PPON} is high, the microcontroller's ALE value passes through to the 87C64's ALE/CE pin. When V_{PPON} is low, ALE/CE can be controlled by the microcontroller's ALECONT signal during programming.

The microcontroller's A_{0-12} outputs are connected to the 87C64's A_{0-12} pins. The EPROM's D_{0-7} are connected to the controller's AD_{0-7} pins. The controller's program-memory read signal, $PSEN$, controls the 87C64's output-enable, OE .

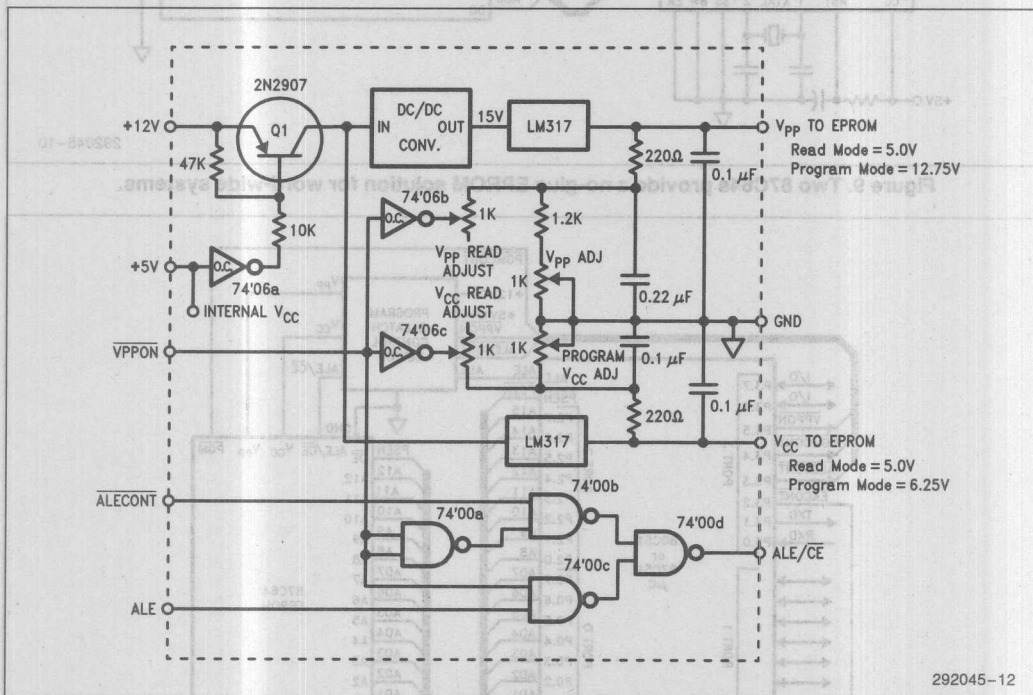


Figure 11. Program- and latch-control circuit for in-system programming.

During programming, the controller brings $\overline{\text{EACONT}}$ high and $\overline{\text{VPPON}}$ low. This allows it to operate from internal code, enables programming voltages on the 87C64's V_{pp} and V_{CC} pins, and switches $\text{ALE}/\overline{\text{CE}}$ control from the controller's ALE to its $\overline{\text{ALECONT}}$. It then inputs data over its serial channel. With $\overline{\text{ALECONT}}$ high, an address is placed on ports 0 and 2. When $\overline{\text{ALECONT}}$ is brought low, the 87C64 internally latches the address. Data read from the serial port is then written to port 0. The 87C64 now has both address and data information. The controller needs only to bring $\overline{\text{PGMCONT}}$ low to program data into the addressed location.

The in-system programming sequence is summarized below.

- 1) Assert $\overline{\text{EACONT}}$. Code is now supplied from the uController's internal program memory.
- 2) Assert $\overline{\text{VPPON}}$. This switches V_{pp} and V_{CC} to their program voltages and allows the controller to manually control ALE via $\overline{\text{ALECONT}}$. $\overline{\text{PGMCONT}}$ and $\overline{\text{ALECONT}}$ are high.
- 3) Input address and data information from Port 3's serial channel. Ports 0 and 2 serve as I/O ports. Place the address on ports 0 and 2. Bring $\overline{\text{ALECONT}}$ low to latch the address into the 87C64.
- 4) Write data information to port 0.
- 5) Bring $\overline{\text{PGMCONT}}$ low to program data into the 87C64. See the 87C64 data sheet for the proper programming algorithm and timing requirements.
- 6) Verify the programmed data. Use the "MOVC A,@A+DPTR" instruction to read EPROM data. The configuration shown in Figure 10 allows the 87C64 to be read at any 8K-byte boundary. This allows the controller to operate using its internal low-memory code and still verify external EPROM mapped at the same locations.
- 7) Repeat this sequence until all EPROM data bytes are programmed and verified.
- 8) When programming is complete, $\overline{\text{VPPON}}$, $\overline{\text{PGMCONT}}$, and $\overline{\text{ALECONT}}$ should be de-asserted. When $\overline{\text{EACONT}} = "0"$, code execution will commence from the 87C64. Duplication of code at identical internal and external memory locations will allow uninterrupted paging between these two memory spaces (see application note AP-284 "Using Page-Addressed EPROMs" for further details).

Care should be taken during system design to ensure that microcontroller and other device inputs can handle elevated voltages supplied by the EPROM during programming. When 6.25V is applied to the 87C64's V_{CC} , its outputs, when "1", will be close to 6.25V.

87C257

The 87C257 is a 256K-bit EPROM organized as 32768 8-bit words. It also contains the equivalent of two 74HCT573 address latches. All address inputs are latched. Figure 12 shows the 87C257's block diagram. To serve high-performance 8-bit microcontrollers, the 87C257 has separate ALE and $\overline{\text{CE}}$ inputs. The 87C257 is pin compatible with the 27C256 (see Figure 4).

The ALE/ V_{pp} input serves as the latch enable during read mode and as the high voltage input during programming. When ALE is high, address information on pins A_0 - A_{14} flows through the latches to the input decoders. If $\overline{\text{CE}}$ is asserted ($\overline{\text{CE}} = V_{IL}$), the EPROM is in its active mode which allows address decoding to begin immediately. If $\overline{\text{CE}}$ is high, the 87C257 is in stand-by mode, but addresses can still be latched. The address latches retain present address-pin values when ALE goes low (ALE = V_{IL}).

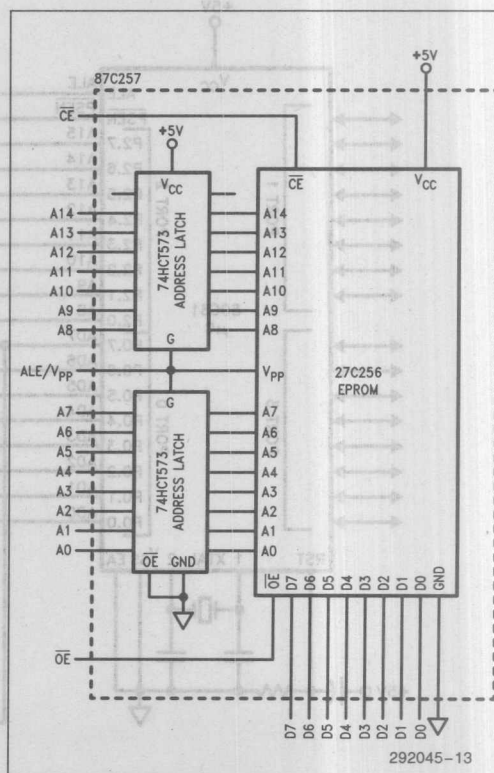


Figure 12. 87C257 functional diagram.

Port 0 pins connect directly to the 87C257's A₀₋₇ and D₀₋₇ pins. Port 2 pins are connected to the 87C257's A₈₋₁₄ and $\overline{\text{CE}}$ pins. Since the 87C257 fills the lower half of the 80C31's program-memory map (0000h – 7FFFh), address line A₁₅ (P2.7) can be connected to the 87C257's $\overline{\text{CE}}$ input. The EPROM is selected whenever A₁₅ is low.

The 80C31's ALE controls an external address latch. When ALE is high, the controller's port 0 and port 2 pins present address information. When low, addresses A₀₋₇ are externally latched. The external latch then supplies the low-address to external memory devices. Since the 87C257 has its own latch, the 80C31's ALE is connected to the 87C257's ALE/VPP (the 87C257's VPP function is internally disabled in read mode).

The 80C31's $\overline{\text{EA}}$ (External Access) pin must be connected to ground when accessing external program memory between addresses 0000h and 0FFFh (the upper address boundary may vary depending on the 8051 version used).

Figure 13. A “no-glue” 80C31/87C257 system.

Two 87C257s + 80C31

8051-family controllers are unique in that two 64K-byte memory spaces can be addressed. These controllers have separate PSEN and RD signals that access program memory (ROM or EPROM) and data memory (RAM and peripheral devices). All system devices see the controller's 16-bit address. Depending on the instruction type, either PSEN or RD is asserted. Although two devices can be memory mapped at identical locations, PSEN and RD determine which will present data.

Figure 14 shows two 87C257s in an 80C31 system. Each 87C257 connects to the 80C31 just as it did in the 87C257 + 80C31 example shown in Figure 13. The only difference is the inverter between A₁₅ and the second 87C257's CE. This inverter allows the second 87C257 to be selected when A₁₅ is high — addresses 8000h – FFFFh.

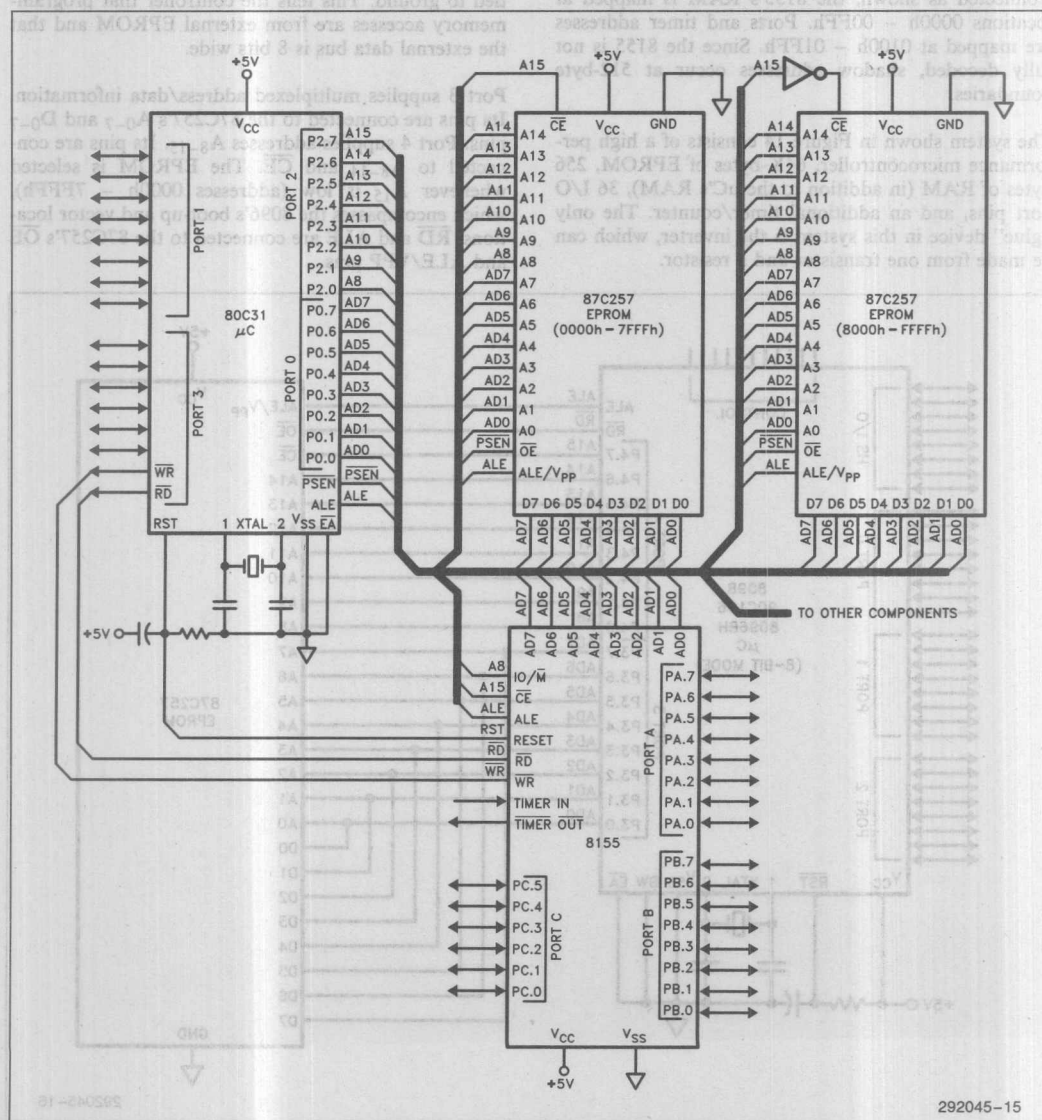


Figure 14. A maximum function, but minimum chip, 80C31 system.

Two 87C257s completely fill the 80C31's program memory space. 64K bytes are still available in the data memory space. A system that requires 64K-bytes of EPROM is probably performing complex I/O tasks. These tasks usually require more RAM than the microcontroller contains. Also, since the 80C31 loses two 8-bit I/O ports when accessing external memory, port reconstruction is desirable.

The 8155 shown in Figure 14 recovers the lost ports (plus 6 additional port pins) and supplies 256 bytes of RAM. In addition, it provides a 14-bit counter/timer. Connected as shown, the 8155's RAM is mapped at locations 0000h - 00FFh. Ports and timer addresses are mapped at 0100h - 01FFh. Since the 8155 is not fully decoded, shadow addresses occur at 512-byte boundaries.

The system shown in Figure 14 consists of a high performance microcontroller, 64K-bytes of EPROM, 256 bytes of RAM (in addition to the uC's RAM), 36 I/O port pins, and an additional timer/counter. The only "glue" device in this system is the inverter, which can be made from one transistor and a resistor.

87C257 + 8096

Intel's 8096-family microcontrollers contain six 8-bit I/O ports, a powerful CPU, and many other high-performance features. 8096BH, 8098, and 80C196 versions also have 8-bit external bus modes that simplify interfaces to 8-bit memories and peripherals. When used in expanded mode, ports 3 and 4 supply the multiplexed address/data bus.

Figure 15 shows a no-glue 8096/87C257 interface. The 8096's EA (External Access) and Buswidth pins are tied to ground. This tells the controller that program-memory accesses are from external EPROM and that the external data bus is 8 bits wide.

Port 3 supplies multiplexed address/data information. Its pins are connected to the 87C257's A₀₋₇ and D₀₋₇ pins. Port 4 supplies addresses A₈₋₁₅. Its pins are connected to A₈₋₁₄ and CE. The EPROM is selected whenever A₁₅ is low (addresses 0000h - 7FFFh), which encompasses the 8096's boot-up and vector locations. RD and ALE are connected to the 87C257's OE and ALE/VPP pins.

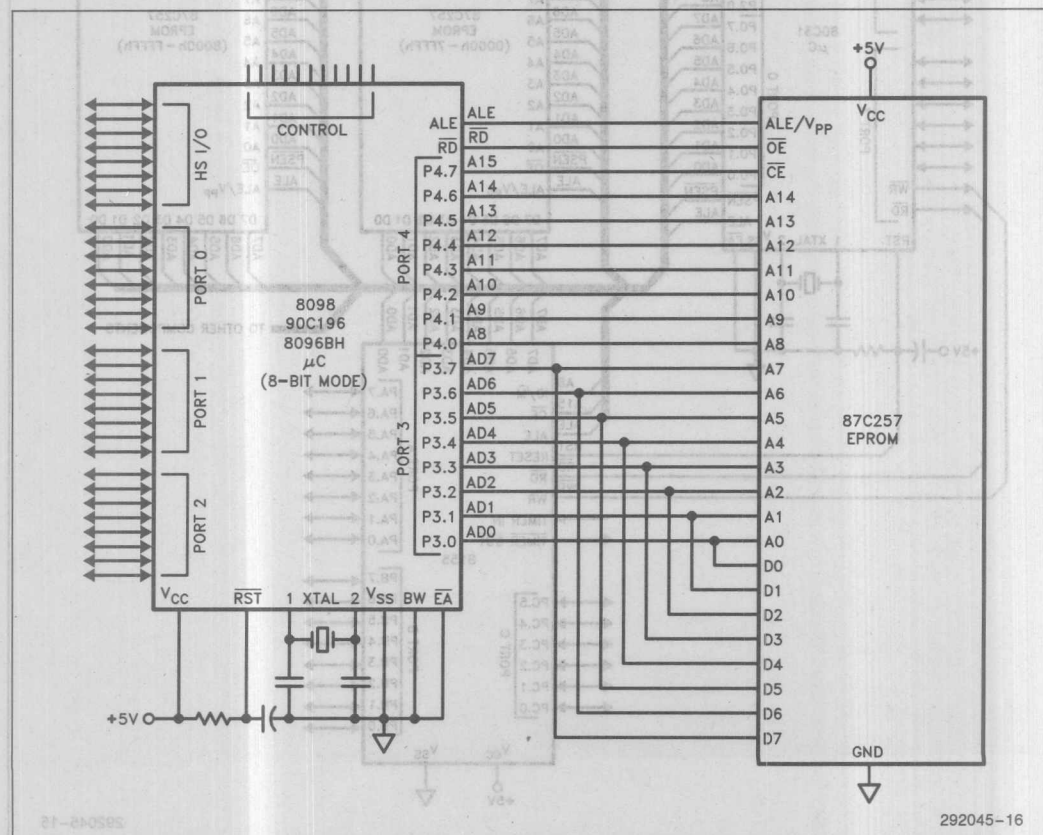


Figure 15. An 87C257 enhances the powerful 8096.

68C257

The microcomputer industry has a standard for memory and peripheral interfaces which dictates chip-enable and output-enable polarities. Customers using non-standard-bus controllers asked Intel to provide a "no-glue" EPROM for their applications — the 68C257.

Like Intel controllers, 68xx-family uCs use multiplexed address/data pins. However, they differ in two significant ways. First, 68xx controllers use high-memory addresses for reset- and interrupt-vectors. Since A15 is high during vector accesses, it can't be connected directly to a standard EPROM's CE — an inverter is required. Second, read and write controls are functions of R/W and E (clock output). Fortunately, EPROMs don't require combinational logic to decode R/W and E. The active-high E output can simply be inverted before connecting it to an EPROM's OE input.

The 32K-byte 68C257 EPROM's inputs contain latches, just like the 87C257. The 68C257 also internally inverts CE and OE. Figure 16 shows the 68C257's block diagram. Figure 17 shows a no-glue 68C257/68xx interface.

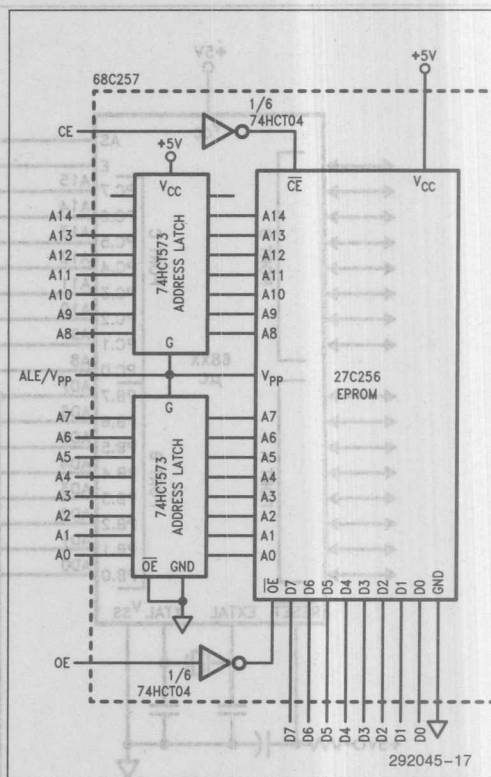


Figure 16. 68C257 functional diagram.

585048-18

Figure 17. The 68C257 is the "no-glue" EPROM for alternate-architecture microcontrollers.

Intel's latched EPROMs reduce chip count and board space, enhance performance, increase reliability, minimize design time, and simplify microcontroller systems. Latched EPROMs are available in popular 64K- and 32K-bit densities, and a version is available that will provide a "no-glue" interface to virtually any microcontroller architecture.

SUMMARY

The best system design is small in size, easy to manufacture, highly reliable, and cost effective. Components that simplify the design process add even more value to the system.

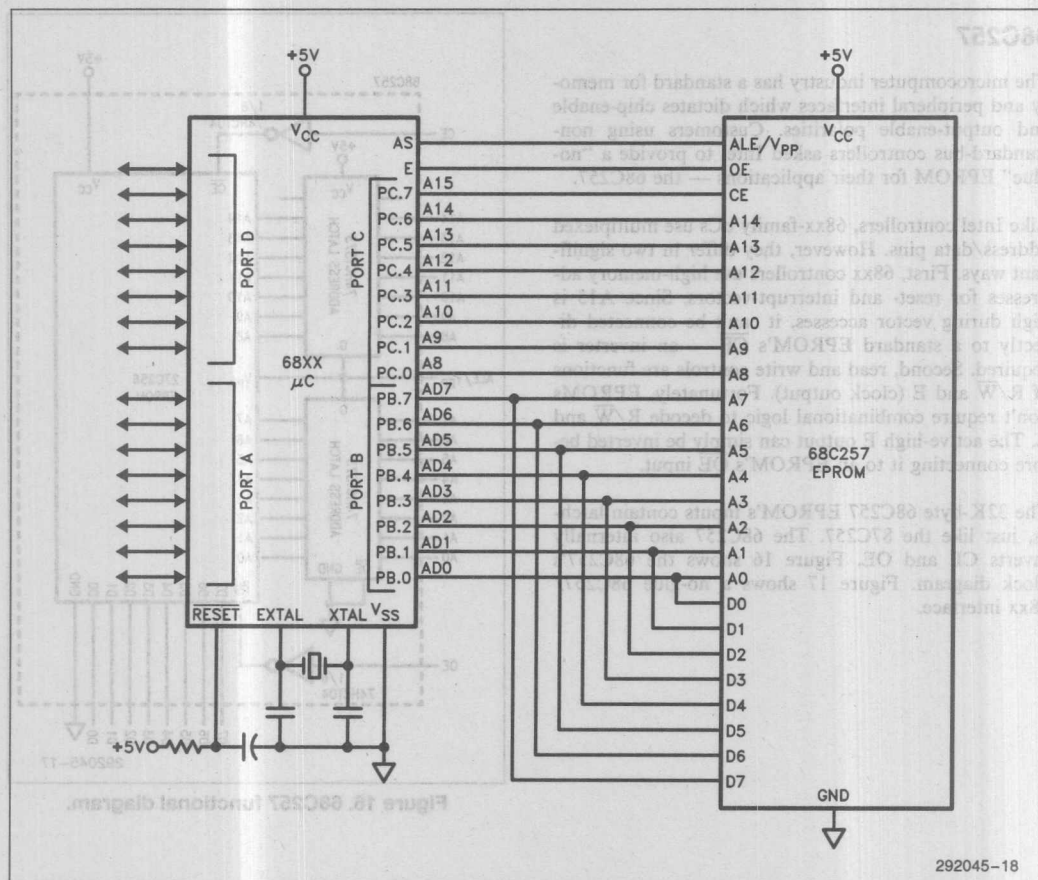


Figure 17. The 68C257 is the "no-glue" EPROM for alternate-architecture microcontrollers.

SUMMARY

The best system design is small in size, easy to manufacture, highly reliable, and cost effective. Components that simplify the design process add even more value to the system.

Intel's latched EPROMs reduce chip count and board space, enhance performance, increase reliability, minimize design time, and simplify microcontroller systems. Latched EPROMs are available in popular 64K- and 256K-bit densities, and a version is available that will provide a "no-glue" interface to virtually any microcontroller architecture.